Detailed Processing Model version 2 (DPMv2)

glaciers cci

Prepared by: Glaciers_cci consortium
Contract: 4000101778/10/I-AM
Name: Glaciers_cci-D3.7_DPMv2
Version: 1.0
Date: 10.10. 2013

Contact:
Frank Paul
Department of Geography
University of Zurich
frank.paul@geo.uzh.ch

Technical Officer:
Stephen Plummer
ESA ESRIN

Document status sheet

Version	Date	Changes	Approval
0.1	24.07. 2012	Initial draft	
0.2	03.08. 2012	Execution summary, introduction, TOC created	
0.3	09.08. 2012	General architecture described	
0.4	24.08. 2012	EC Altimetry TOC expanded and first compilation	
0.5	19.10. 2012	Glacier area, EC DEM differencing, Velocity sections reviewed	
0.6	13.02. 2013	Comments from technical officer integrated	
0.7	13.03. 2012	Review from consortium integrated	
0.8	25.03. 2013	Revised version	
0.9	20.08. 2013	New comments from TO integrated	
1.0	10.10. 2013	Desription of support modules for altimetry added	F. Paul

The work described in this report was done under the ESA contract 4000101778/10/I-AM. Responsibility for the contents resides with the authors that prepared it.

Author team:
Francesca Ticconi, Andrew Shepherd (SEEL), Frank Paul (GIUZ), Andreas Kääb, Christopher Nuth (GUIO), Thomas Nagler, Helmut Rott, Kilian Scharrer (Enveo), Tazio Strozzi, Frey Othmar (Gamma)

Glaciers_cci Technical Officer at ESA:
Stephen Plummer

Table of Contents

1. Executive Summary

According to the Statement of Work, the Detailed Processing Model (DPM) should provide a structured description of the computer algorithms to be applied for product generation. We describe them in this document for each of the five products (1) glacier area, elevation changes from (2) DEM differencing and (3) altimetry, and velocity from (4) optical and (5) microwave sensors separately. We have developed a data processing scheme that is based on a modular concept with macro-modules on the product level and sub-modules for the key processing steps. In this document we do not identify how a specific processing step or module is currently implemented in (commercial) software products. The focus is on a functional description of the algorithms and the steps to be performed to allow a software-independent implementation. Iinput and output data and their preparation are described in the IODD.

The five products have very different demands in regard to user interaction, computational resources, algorithm complexity and dependence on already available software products. In this regard, the descriptions of the individual modules vary strongly. While the glacier area product has only four modules performing specific operations on complete files, product (3) uses a large number of modules with complex calculations and dependencies. Also product (2) is rather light in regard to required computer resources, but operator interaction is high and requires several processing steps outside the modules, for example using a graphical user interface (GUI) to visualize and modify data. The two velocity products (4 and 5) finally, are between the two extremes. Many of the modules described here require processing with specialized software (in general commercial products), but others are rather light and require operator decisions to achieve highest product accuracy.

2. Introduction

2.1 Purpose and scope

This is the Detailed Processing Model version 1 (DPMv1) of the Glaciers_cci project. It is the seventh deliverable of Task 2 (D2.7). According to the Statement of Work, the DPMv1 provides a compact and informal but structured high-level description of the computer programming algorithms. It shall provide an environment-independent description of

- the key principles of the algorithm
- the top-down decomposition of the software into its components
- a detailed list and description of the variables used in the mathematical equations
- the computer program in pseudo-code.

Following these requirements, the description of the algorithms for ECV production is arranged in a modular form. The document starts with a general description of the software architecture and the macro-module concept. It then describes in detail for each of the products

- glacier area,
- elevation change (from DEM differencing and altimetry),
- velocity (from optical and microwave sensors)

the general processing workflow and all modules related to data processing (the input and output interfaces are described in the IODD). Each module has a general description, a definition and detailed description of the objects to be computed, a definition of variables (input, internal, and output), an overview on model equations (if any) along with a logical flow diagram and the pseudo code (see section 2.2). All modules are described independently to fully consider product specific differences. However, from a system engineering perspective their functionality is often very similar..

It has to be noted that there are some differences in the general set-up of the processing for the various products. For example, the glacier area product is created by short scripts that are operated from a graphical user interface (GUI) provided by a commercial software product, while the altimetry product can be created by starting a complex program code at the command line. While for the latter the calculations are largely based on variables, the variables for the former are files or even folders (vector, raster) that are directly manipulated with the scripts. Similar differences apply for the input and output data that are satellite images (GeoTiff) for the area product and numbers in columnar format (ASCII) in the case of altimetry. Despite these principle differences, we have tried to harmonize the description of the modules for all products to have a better overview on potential similarities. In particular for the velocity product it is possible that certain modules can be shared as key processing algorithms are rather similar. This is not the case for the elevation change product, which requires a completely different set-up for the respective processing system.

2.2 General architecture description

The Glaciers_cci ECV production system is a combination of various tools conceived to produce the four products glacier area, elevation changes from altimetry and DEM differencing, and velocity (jointly from optical and microwave data). The system has been designed with the aim to have a modular structure and consists of five different macro-modules containing modules with each of them implementing a given functionality, using EO data and the required further data as input, and providing the three main products as output (Fig. 2.1). With this characteristic, the architecture is more flexible and the updating is easier to perform. The flexibility aspect is strongly related to the possibility to add further modules, whereas the updating consists of the possibility to modify or change the computation of the output data. The input and output interfaces shown in Fig. 2.1 prepare the input data (mostly format conversions) allowing the correct functioning of each macro-module and make the output data of the processing system available in the format described in the PSD. They are described along with the format of the respective datasets in the IODD (Glaciers_cci, 2013).

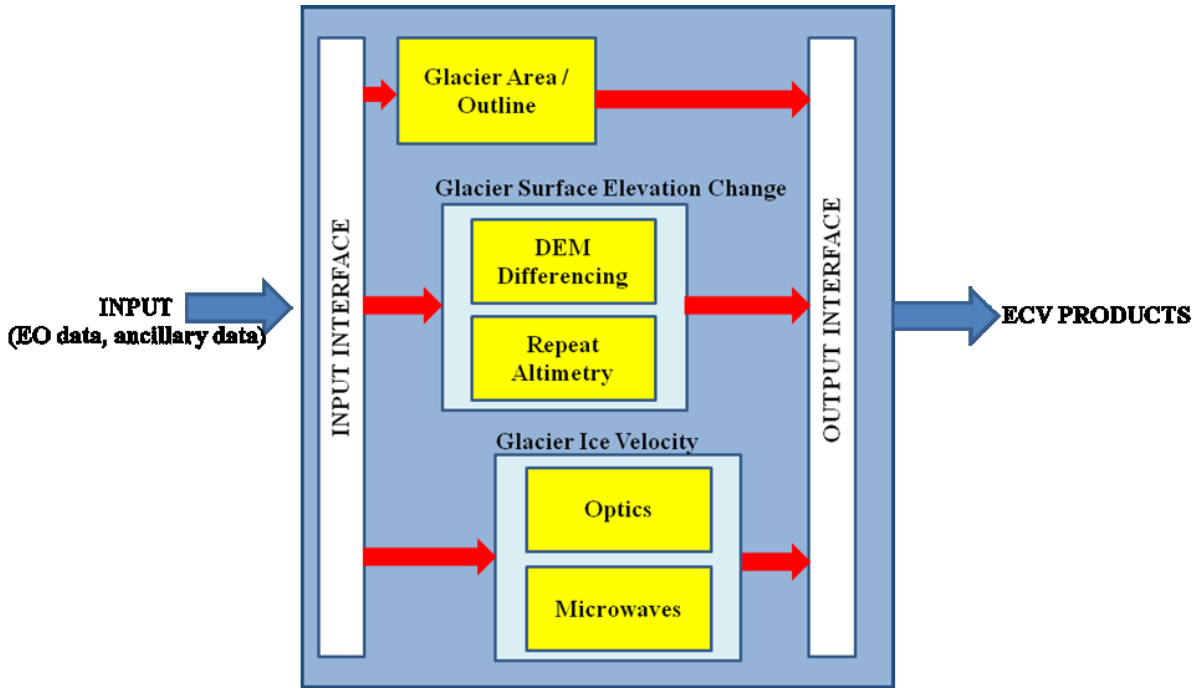

Fig. 2.1: Architecture of the Glaciers_cci ECV production system.

For the elevation change product two different techniques are considered: DEM differencing and repeat altimetry. As a result, the system will have two macro-modules (illustrated in Fig. 2.2) with a very different functionality. The velocity product is derived from two different EO data sources as well, optical and microwave, but after input datasets have been prepared, the main processing is rather similar. This also applies to some potential support modules (e.g. for coordinate transformations) that might be shared (Fig. 2.3).

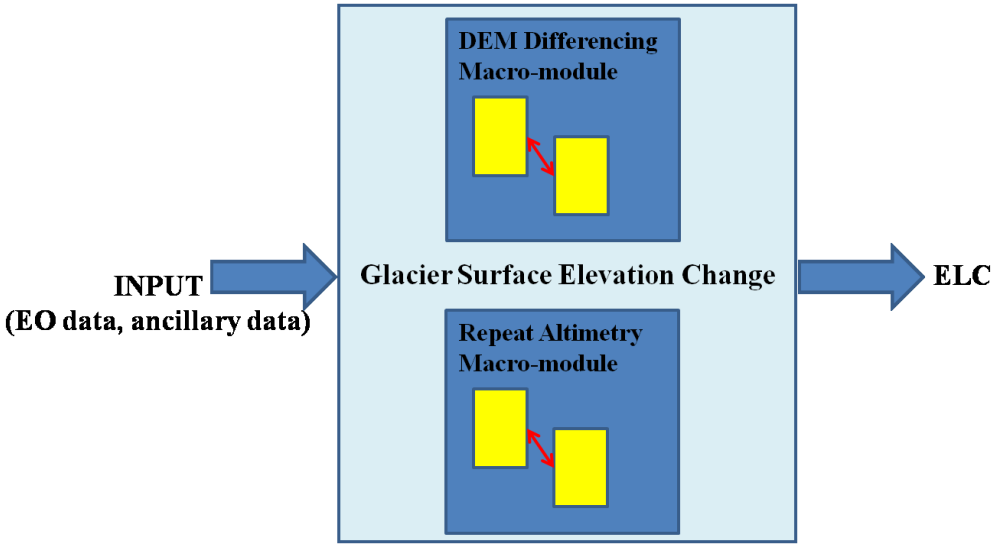

Fig. 2.2: Macro-modules for the surface elevation change product.

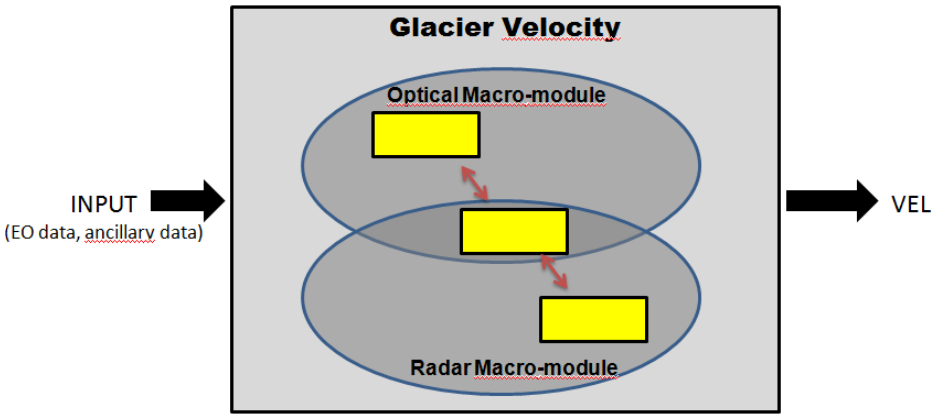

Fig. 2.3: Macro-modules for the glacier velocity product. The overlapping spheres for optical and microwave data represent the potential for joint modules.

In the following sections we provide for each of the macro-modules:
- a general description of the macro-module functionality;
- a general block diagram showing the set of modules composing each macro-module and the data flow between them;
- the time sequence defining the operational flow within each macro-module.

Afterwards we provide for each of the processing modules:
- a general description to illustrate the object to compute;
- a detailed list of the input/output and internal parameters;
- a more detailed description of the mathematical background for each module (when necessary);
- a logical flow diagram;
- the pseudo code for each module.

3. Glacier area macro-module

3.1 General macro-module description

The glacier area macro-module is part of an overall data processing workflow (shown in Fig. 3.1) consisting of several elements that are not part of the macro-module structure, like all the user interactions that are based on a Graphical User Interface (GUI). This GUI related work includes pre-processing of the input datasets (e.g. creation of RGB composites), manual editing of glacier outlines (e.g. correction of water, debris and shadow), and other interactive steps. The four modules forming the glacier area macro-module are part of the main and post-processing stage of the general workflow.

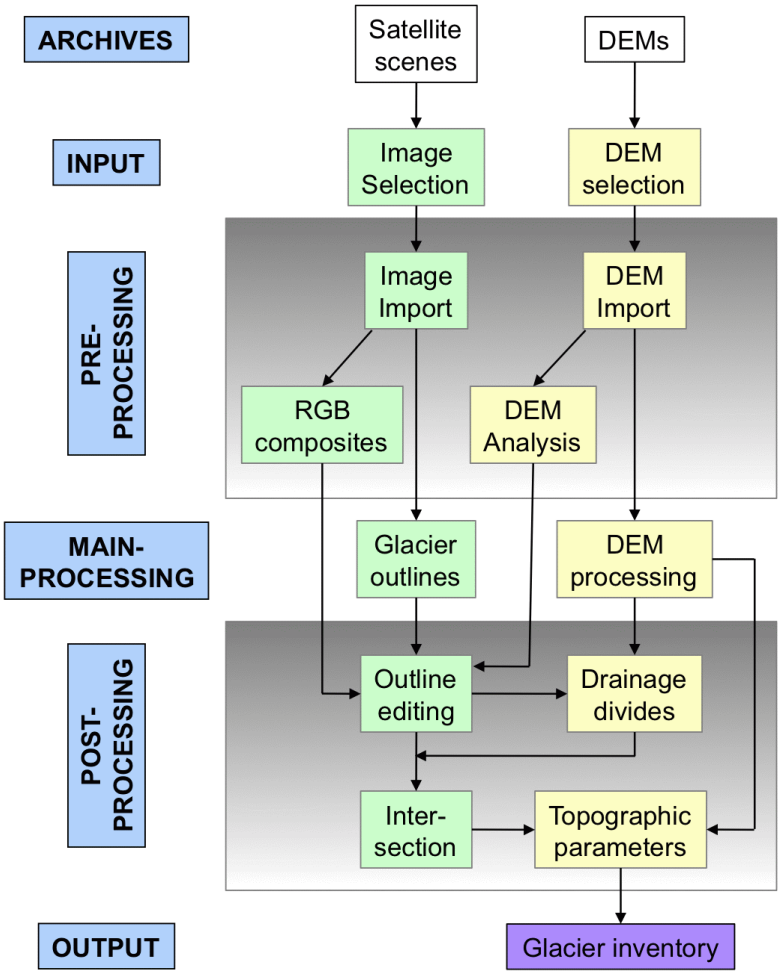

Fig. 3.1: The general workflow for the glacier area product. The processing steps are indicated as well.

The schematic structure within the macro-module is illustrated in Fig. 3.2. In short, the module computes from a satellite scene and a DEM (input) glacier outlines, drainage divides and topographic parameters for each glacier entity (output). The four modules are:

- GAM1 (Outlines): calculates raw glacier outlines from the input satellite image;

- GAM2 (DEM): calculates several DEM derivatives from the input DEM;
- GAM3 (Divides): derives drainage divides for each glacier;
- GAM4 (Topo): digitally intersects the glacier outlines with the drainage divides and derives topographic parameters for each glacier.
- GUI1 (Threshold): interactive selection of two threshold values for glacier classification
- GUI2 (Outl-Edi): correction of wrongly classified regions (e.g. water, clouds, debris)
- GUI3 (Basin-Edi): correction of the raw drainage basins to glacier specific ones

In principle, each glacier area module (GAM) is a short text script with a series of instructions that run with a specific software. The output of GAM1 and GAM3 has to be edited outside the module before serving as an input for GAM3 and GAM4, respectively. Therefore, user interaction is a required element of the entire processing chain.

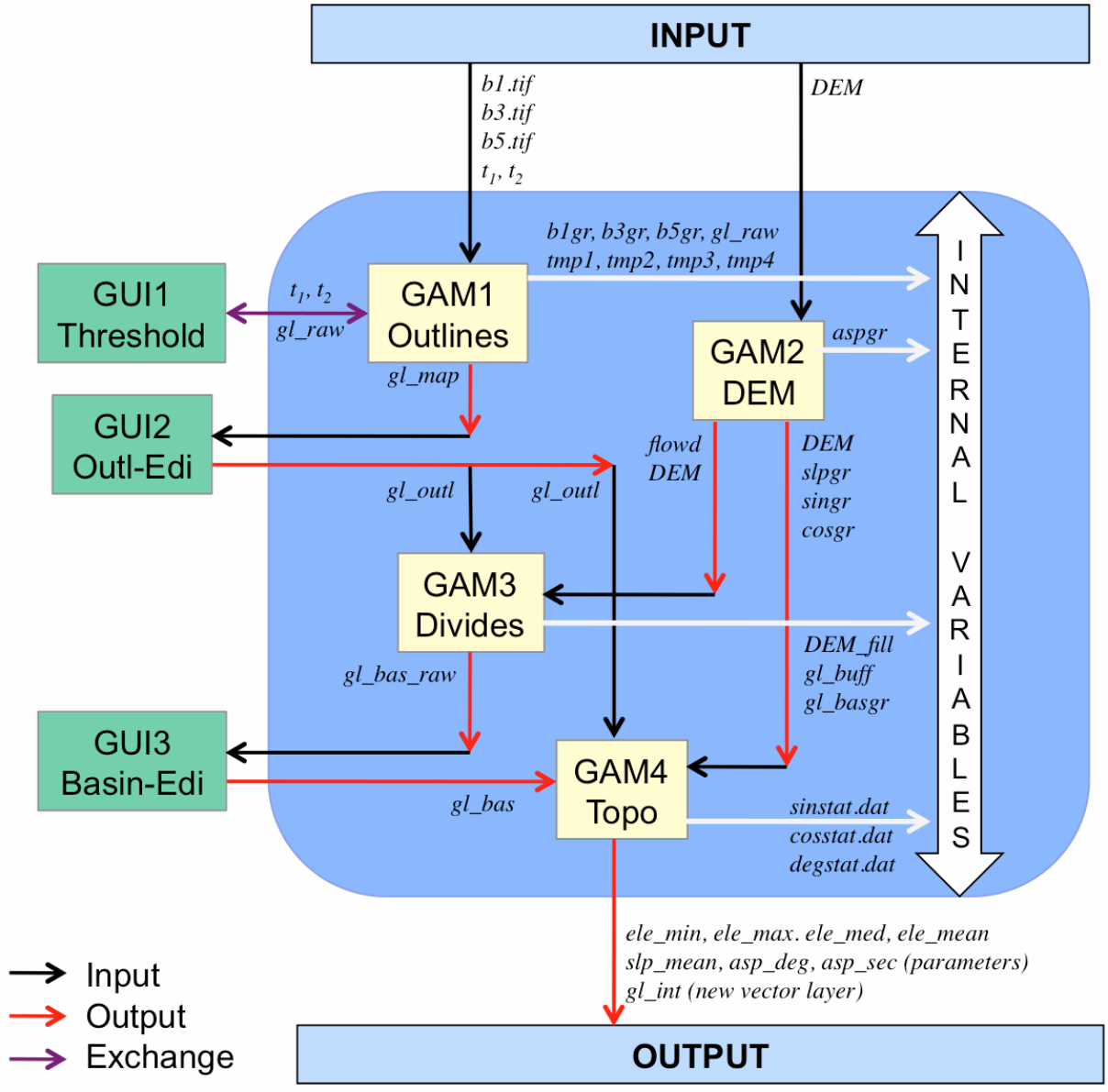

Fig. 3.2: Structure of the glacier area macro-module.

3.2 Module description

3.2.1 GAM1: Outlines module

General description

The outline module (GAM1) calculates raw glacier outlines from the satellite image.

Definition and detailed description of the objects to be computed

Three bands of the satellite image are imported (input) as a geocoded GeoTiff (image format) and are exported (output) as one data layer comprising all glacier outlines (vector format). Polygons refering to regions outside of glaciers get a no-data code (e.g. -9999) and glacier polygons are getting another code (e.g. 0) and an unique identifier (ID) in the attribute table.

Definition of variables

All data files and vector layers are treated as variables. Real variables (i.e. scalar values) are marked by a (v).

Input variables:

- *b1.tif* = satellite image band in the blue part of the spectrum;
- *b3.tif* = satellite image band in the red part of the spectrum;
- *b5.tif* = satellite image band in the shortwave-infrared part of the spectrum;
- t_1 = threshold for the band ratio (v);
- t_2 = threshold for the blue band (v).

Internal variables:

- *b1gr* = band1.tif in internal grid format;
- *b3gr* = band3.tif in internal grid format;
- *b5gr* = band5.tif in internal grid format;
- *tmp1* = resulting temporary grid after threshold t_1 is applied;
- *tmp2* = resulting temporary grid after threshold t_2 is applied;
- *tmp3* = resulting temporary grid after median filter is applied;
- *tmp4* = resulting temporary grid with no-data (-9999) assigned to non-glacier areas.

Output variables:

- *gl_raw* = vector outlines created from *tmp4* by raster vector conversion.

Model equations and logical flow diagram

Glaciers are classified when the band ratio (*b3gr*/*b5gr*) is larger than t_1 and when additionally *b1gr* is larger than t_2. This algorithm has been widely used in the literature (e.g. Paul and Kääb, 2005) and was selected as the key algorithm for Glaciers_cci in the PVASR (Glaciers_cci, 2012a). Details of the processing are illustrated in the ATBD (Glaciers_cci, 2012b). The logical form of the processing equation is:

$$\text{IF } [(b3gr/b5gr) > t_1] \text{ and } (b1gr > t_2) \text{ THEN pixel} = 0 \text{ ELSE pixel} = 255$$

The term 'pixel' refers to each pixel (or cell) of the resulting raster dataset. The result of this operation is stored in the file *tmp2*. The final output after filtering and raster-vector conversion (*gl_raw*) has to be visualised outside the module to decide whether the applied threshold values t_1 and t_2 produce good results. If yes, the output is renamed *gl_map* and can be used for processing in the next module (GAM3), if not the threshold values have to be corrected and the algorithm must be applied again. Once a successful result is generated, all temporary grids (*tmp1*, ..., *tmp4*) are removed and the file *gl_raw* is either to be removed or renamed (if it should be used for comparison with the next result), e.g. to *gl_rawxy* with *xy* being t_1 * 10. The logical flow diagram of the module and its interaction with the outside processing step (adjustment of the thresholds) are illustrated in Fig. 3.3.

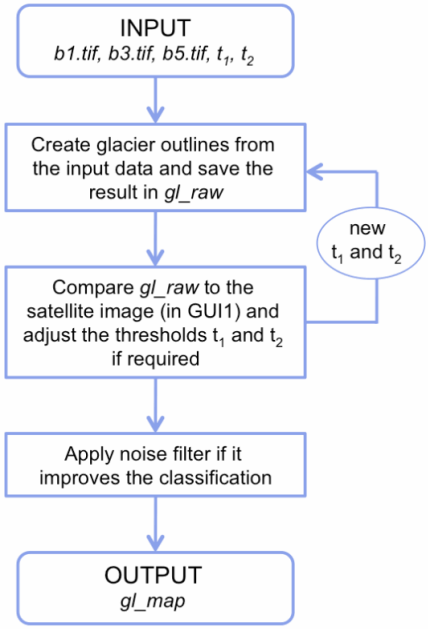

Fig. 3.3: Logical flow diagram of the outlines module GAM1.

Pseudo Code

The pseudo code for the module is reported below.

1. *Convert the three input image bands band1.tif, band3.tif, and band5.tif to the internal raster files b1gr, b3gr, and b5gr, respectively.*
2. *Compute the band ratio (b3gr / b5gr) and assign 0 to all cells where the ratio is > t_1 and 255 to all cells where the ratio is < t_1,*
3. *Save the result of step 2 in tmp1.*
4. *Assign 255 to all cells with a value of 0 in tmp1 when b1gr < t2*
5. *Save the result of step 4 in tmp2*
6. *Apply a 3 by 3 median filter (to reduce noise) to tmp2 and save the result in tmp3*
7. *Set all cell values = 255 in tmp3 to no data and save the result in tmp4*
8. *Convert tmp4 to vector format and save the result in gl_raw*
9. *Analyze gl_raw in the GUI1 and decide if threshold t_1 and/or t_2 need to be adjusted*
10. *If yes, change the thresholds, remove all temporary files and continue with step 2*
11. *If no, save the result gl_map under a more appropriate file name.*

3.2.2 GAM2: DEM module

General description
The DEM module (GAM2) calculates several DEM derivatives from the input DEM.

Definition and detailed description of the objects to be computed
The input raster layer (DEM) is used to derive further raster layers (DEM derivatives) that are used as an input in GAM3 (flow direction grid) and GAM4 (slope, sine and cosine grids). While the flow direction grid is 8bit integer, the other grids store real values in each cell.

Definition of variables
All data files and vector layers are also treated as variables. Real variables (i.e. scalar values) are marked by a (v).

Input variables:
- *DEM* = Digital elevation model covering the study region (grid).

Internal variables:
- *aspgr* = Aspect of the terrain, used to calculate the sine and cosine grids.

Output variables:
- *slpgr* = Slope of the terrain;
- *singr* = Sine of *aspgr*;
- *cosgr* = Cosine of *aspgr*;
- *flowd* = Flow direction grid.

Model equations and logical flow diagram
The module calculates a slope, aspect and flow direction grid (*slpgr*, *aspgr*, *flowd*) from the DEM using software specific built-in scripts. The aspect grid (*aspgr*) is then used to derive a sine and a cosine grid (*singr*, *cosgr*).

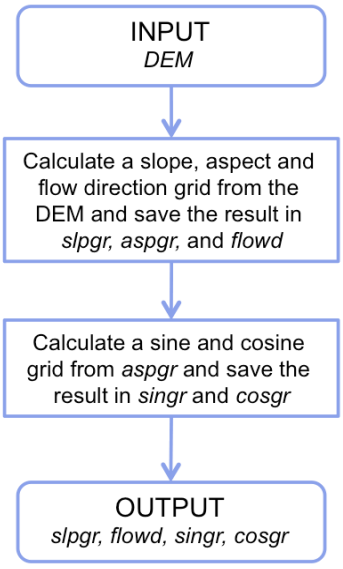

Fig. 3.4: Logical flow diagram of the DEM module GAM2.

Pseudo Code

1. *Compute a slope, aspect and flow direction grid from the input DEM*
2. *Save the result of step 1 in the files slpgr, aspgr, and flowd*
3. *Calculate a sine and a cosine grid from aspgr*
4. *Save the result of step 3 in the files singr and cosgr*

3.2.3 GAM3: Divides module

General description

The divides module (GAM3) combines the output from GAM1 (after editing in GUI2, see 3.2.6) with the output from GAM2 (*flowd*) to derive drainage divides for each glacier. This module is basically a logical arrangement of commands to be performed by an operator with GIS software providing a GUI (for display and running the commands). There are several solutions for determining glacier divides available in the literature, partly differing due to other software used for processing. Hence, the general structure described here might not work 1:1 for other software products. A recent study by Kienholz et al. (2013) has determined drainage divides for all glaciers globally. With the divides and the algorithm becoming available, the GAM3 module described here might become obsolete.

Definition and detailed description of the objects to be computed

The calculation is based on a script presented by Bolch et al. (2010). It requires a DEM (grid), a flow direction grid and the glacier outlines (vector) as an input and computes glacier drainage divides (vector) from watershed analysis and buffered glacier extents.

Definition of variables

All data files and vector layers are also treated as variables. Real variables (i.e. scalar values) are marked by a (v).

Input variables:

- *DEM* = Digital elevation model covering the study region (grid);
- *gl_outl* = Glacier map (vector) from GAM1 after manual editing in GUI2
- *flowd* = Flow direction grid derived from the *DEM* in GAM2.

Internal variables:

- *DEM_fill* = DEM with all sinks (local depressions) filled
- *gl_buff* = buffered glacier outlines from *gl_outl*
- *gl_basgr* = derived hydrologic catchments in raster format

Output variables:

- *gl_bas_raw* = hydrologic catchments in vector format

Model equations and logical flow diagram

The module calculates a buffer around the glacier outlines, extracts the DEM within this buffer and performs a watershed analysis on this clipped DEM. The resulting glacier basins are manually edited afterwards and then digitally intersected with the glacier outlines.

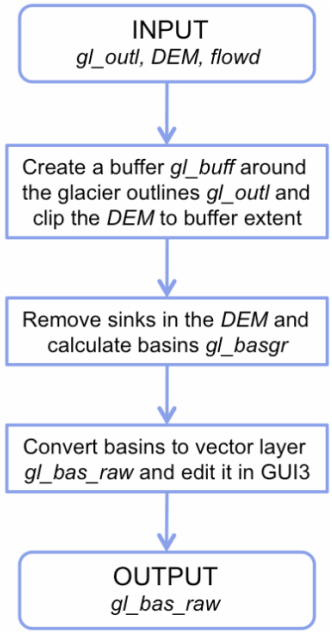

Fig. 3.5: Logical flow diagram of the Divides module GAM3.

Pseudo Code

1. *Create a buffer around the existing glacier polygons*
2. *The buffer distance varies according to the glacier sizes. Suitable values are between 1 and 1.5 km*
3. *Clip the DEM to the buffer extent*
4. *Remove all sinks from DEM*
5. *Calculate glacier basins*
6. *Convert raster basins to vector layer*

3.2.4 GAM4: Topo module

General description

The Topo module (GAM4) digitally intersects the corrected glacier outlines from GUI2 (see 3.2.6) with the divides from GAM3 (after editing in GUI3) and derives topographic parameters for each glacier using the GAM2 output as an input.

Definition and detailed description of the objects to be computed

The module requires two vector datasets (the edited glacier outlines *gl_outl* and the edited drainage divides *gl_bas*) and four grids from GAM2 (*DEM*, *slpgr*, *singr*, *cosgr*) as an input. It derives for each glacier the respective values from zone statistics with the glacier extent defining a zone and the input grids providing the values. The mean sine and cosine values have to be converted into correct mean aspect values (in degrees and as a sector using the eight cardinal directions). For illustration on how this conversion might look like, we show the code of a Fortran program that has been used previously. The calculated scalar numbers (floating points and integer) and characters are digitally joined with the attribute table of each glacier using build-in software scripts.

Definition of variables
All data files and vector layers are also treated as variables. Real variables (i.e. scalar values) are marked by a (v).

Input variables:

- *DEM* = Digital elevation model covering the study region (grid);
- *slpgr* = slope of the terrain;
- *singr* = sine of *aspgr*;
- *cosgr* = cosine of *aspgr*;
- *gl_outl* = corrected glacier outlines (from GUI2);
- *gl_bas* = corrected glacier basins (from GUI3).

Internal variables:

- *sinstat.dat* = a text file containing glacier ID and mean sine in two columns
- *cosstat.dat* = a text file containing glacier ID and mean cosine in two columns
- *degstat.dat* = a text file containing glacier ID and mean aspect in degrees (floating point) and as a sector from 1 to 8 (integer)

Output variables:

- *gl_int* = vector layer with individual glaciers
- *ele_min* = minimum elevation of the respective glacier (v);
- *ele_max* = maximum elevation of the respective glacier (v);
- *ele_med* = median elevation of the respective glacier (v);
- *ele_mean* = mean elevation of the respective glacier (v);
- *slp_mean* = mean slope (v);
- *asp_deg* = mean aspect in degrees (v);
- *asp_sec* = mean aspect sector from 1 to 8 (with 1=N, 2=NE, etc.) (v).

Model equations and logical flow diagram
At first the glacier outlines (*gl_outl*) are digitally intersected with the drainage divides (*basin*) to create individual glaciers out of the glacier complexes. This is a built-in command specific of the software used. After topology is rebuilt, each glacier entity has a unique ID that is further used to identify the glacier. The calculation of the slope, aspect sine and cosine grids as well as the zonal statistics are also based on built-in software scripts that are not further detailed here). The Fortran code to derive the correct mean aspect from the zonal mean values of the sine and cosine grids is published in Paul (2007) and can also be derived differently. The key equations of the Fortran script used are:

```
n=(ATAN2(x,y))*f
d=MOD(360.+n,360.)
s=(MOD(NINT(d/45),8))+1
```

with n = supporting variable, x = mean sine, y = mean cosine, f = 180 / π, d = mean aspect in degrees, s = mean aspect sector, ATAN2 = arctangent function (in radians), NINT = Integer function, and MOD= modulo function.

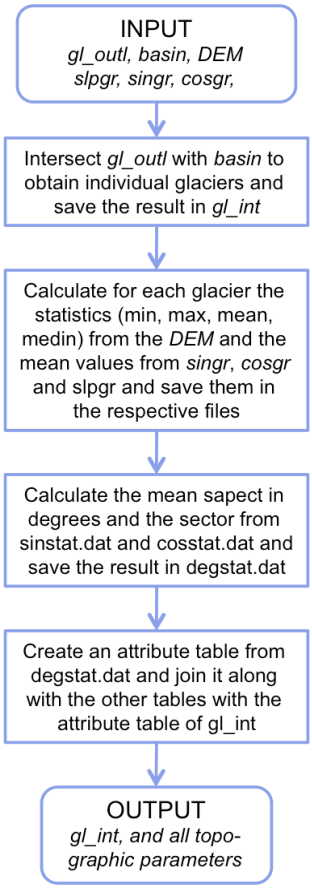

Fig. 3.6: Logical flow diagram of the topo module GAM4.

Pseudo Code
The pseudo code for the module is reported below.

1. *Intersect the glacier outlines (gl_outl) from GAM1 with the drainage divides (gl_bas) from GAM3 to create gl_int*
2. *Select the item identifying the ID of each glacier for later identification*
3. *Calculate zone statistics from the DEM, and mean values from the slope grid and the sine and cosine grids within each glacier entity*
4. *Add the former to the attribute table and export the sine and cosine values to ASCII (sinstat.dat & cosstat.dat)*
5. *Convert mean sine and cosine values to the correct aspect sector and export as ASCII*
6. *Join the mean aspect value and sector with the attribute table of the respective glacier*

3.2.5 GUI1: Threshold

General description
GUI1 supports the operator-based decisions on the thresholds t_1 and t_2 to be made in GAM1.

Definition and detailed description of the GUI to be applied
GAM1 generates glacier outlines from pre-defined thresholds for the band ratio and the blue (Landsat) or green (ASTER, SPOT) band. These outlines are overlain with the original

satellite image and their suitability is analyzed (preferably in regions of shadow). Depending on the deviation (too little/much ice mapped) the thresholds are lowered/increased and GAM1 is run again with new values of the thresholds until a best possible match is achieved. The governing principle for threshold selection is to minimize the workload for post-processing, which means to obtain glaciers in cast shadow mapped as well as possible (debris has to be corrected anyway). In the general case, noise increases when the selected thersholds are too low. Depending on the experience of the analyst, three to four iterations with GAM1 are sufficient to find optimal thresholds. The finally selected threshold values t_1 and t_2 should be stored with the processing script.

Definition of variables
The two threshold values t_1 for the band ratio and t_2 for the blue (or green) band are iteratively selected to produce the best results with GAM1 (no internal variables are required).

Input variables:
- t_1, t_2 = Threshold values (scalar).

Output variables:
- t_1, t_2 = Threshold values (scalar).

Software to be used
The GUI1 software must be able to display vector data (shapefiles) on top of the original (or pre-processed) raster data (GeoTiff) and should have a fast zoom in/out function.

3.2.6 GUI2: Outl-Edi

General description
GUI2 is used to edit the outlines generated by GAM1.

Definition and detailed description of the GUI to be applied
The outlines produced by GAM1 have errors that require correction. The most common are related to three types: (1) commission errors (water, clouds), (2) omission errors (debris, clouds and shadow), and (3) correctly mapped elements that are not a glacier (seasonal snow, sea ice and ice bergs). Gross errors like wrongly mapped lakes, rivers or sea ice (often consisting of hundreds of individual polygons) can be easily selected and removed in the vector domain. Objects in contact with a glacier (e.g. sea ice at the terminus, shadow, or a cloud hiding the perimeter) need more careful correction. The editing of debris-covered glaciers and seasonal snow is the most difficult issue since both have identification problems. Getting these regions correctly mapped is actually the most difficult and time-consuming part of the work.

Definition of variables
The raw glacier outlines of GAM1 serve as an input and corrected outlines are generated as an output (no internal variables are required).

Input variables:
- *gl_map* = Raw glacier outlines (vector)

Output variables:

- *gl_outl* = Corrected glacier outlines (vector)

Software to be used

The GUI2 software must be able to display vector data (shapefiles) on top of the original (or pre-processed) raster data (GeoTiff), should have a fast zoom in/out function, and the capability to edit vector data (reshape, select, add, delete of points, lines and polygons).

3.2.7 GUI3: Basin-Edi

General description

GUI3 is used to edit the basins generated by GAM3.

Definition and detailed description of the GUI to be applied

The drainage divides and basins created by GAM3 have errors, e.g. due to DEM artefacts or in flat regions where flow separation is difficult. These outlines have thus to be corrected, basically by merging numerous smaller polygons to a smaller number of larger polygons representing the catchment area of a specific glacier. If glaciers are already separated by natural drainage divides (e.g. rock outcrops) a basin is not required (it is however useful for other purposes such as joint assignment of IDs). A colour-coded flowdirection grid and contour lines (both derived from the DEM in GAM2) are most helpful in interpreting and assigning the correct basins.

Definition of variables

The raw glacier outlines of GAM1 serve as an input and corrected outlines are generated as an output (no internal variables are required).

Input variables:

- *gl_bas_raw* = raw basins (vector)
- *flowd* = *flowdirection grid*
- *DEM* = *used to derive contour lines*

Internal variables:

- *gl_bas_ed1* = basins without a glacier removed (from *gl_bas_raw*)
- *gl_bas_ed2* = merged glacier basins (from *gl_bas_ed1*)

Output variables:

- *gl_bas* = final divides (edited with flowd) for intersection with *gl_outl*

Software to be used

The GUI3 software must be able to display vector data (shapefiles) on top of the original (or pre-processed) raster data (GeoTiff), should have a fast zoom in/out function, and the capability to edit vector data (reshape, select, add, delete of points, lines and polygons).

4. Elevation change from DEM differencing

4.1 General macro-module description

The macro-model for glacier elevation changes from DEM differencing is shown schematically in Fig. 4.1. The import module and the input datasets are described in the IODD (Glaciers_cci, 2013) as it is actually an interface for data selection by the user. At a minimum it requires that the two DEMs overlap spatially and with glaciers. This input is forwarded to the pre-processing module where the bulk of the computations are performed. After import, the DEMs must be resampled and subtracted before detection of mis-registrations can occur. The algorithm determined in the PVASR (Glaciers_cci, 2012b) is used to check and correct for mis-alignments. After co-registration, one of the DEMs must be adjusted, and then checked whether there is still mis-alignment. This is an iterative process requiring two to three iterations (represented by the dotted line in Fig.4.1) using either the Resampling Module in the pre-processing for iteration or the main-processing for the final differencing.

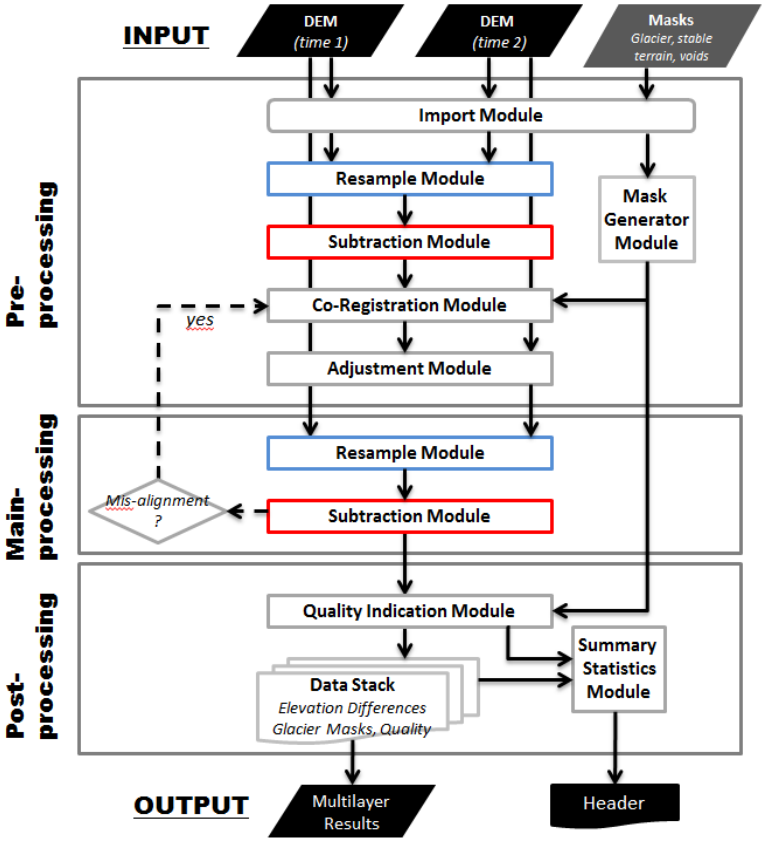

Fig. 4.1: Structure of the glacier elevation change by DEM differencing macro module.

Once mis-alignments are removed, the original slave DEM is adjusted (horizontally and vertically) before resampling and differencing takes place. The post-processing entails quality indication, statistics generation and export into final output format. For the quality indication,

many masks may potentially be exported, including void masks, glacier yes/no masks, and quality masks based upon the individual score masks provided for each DEM. The specific input and output formats are described in the IODD.

4.2 Module description

4.2.1 Resample module

General description and definition of the objects to be computed
This module resamples one DEM grid (slave dataset) to match that of the other DEM (master). It adjusts the cell size of both DEMs to a common value. The selection of the resampling procedure depends on the direction of the sampling, i.e. whether a low or high resolution DEM is resampled to a higher or lower resolution, respectively. Both DEMs will have the same cell size afterwards.

Definition of variables

Input variables:

- *vec_x1,vec_ y1, mtx_dem1*
- *vec_x2,vec_ y2, mtx_dem2 OR*
- *vec_x2_adj,vec_ y2_adj, mtx_dem2_adj*
- *mtx_score1, mtx_void1, mtx_score2, mtx_void2*
- *const_time1, const_time2*

Internal variables:

- *p1, p2* : pixel sizes for *mtx_dem1* and *mtx_dem2*, respectively
- *r:* ratio of the larger to smaller pixel size

Output variables:

- *temp_vec_x:* the vector denoting the common x-coordinates for each grid
- *temp_vec_y:* the vector denoting the common y-coordinates for each grid
- *mtx_dem1, mtx_score1, mtx_void1*
- *temp_mtx_dem2, temp_mtx_score2,temp_mtx_void2:* resampled to coincident grid with mtx_dem1.

Model equations and logical flow diagram
In cases where one pixel size is larger than the other, down-sampling to the largest pixel size is performed using mean block filters. The one grid (smallest) is then interpolated into the grid of the larger pixel DEM such that the grids are coincident. Either bilinear or bicubic interpolations can be used as decided by the analyst. Moreover, void masks and correlation masks, if available, are also resampled following similar procedures as that used for the DEMs. The resampling procedures described here are standard commands in all suitable software packages and are thus not described here in more detail.

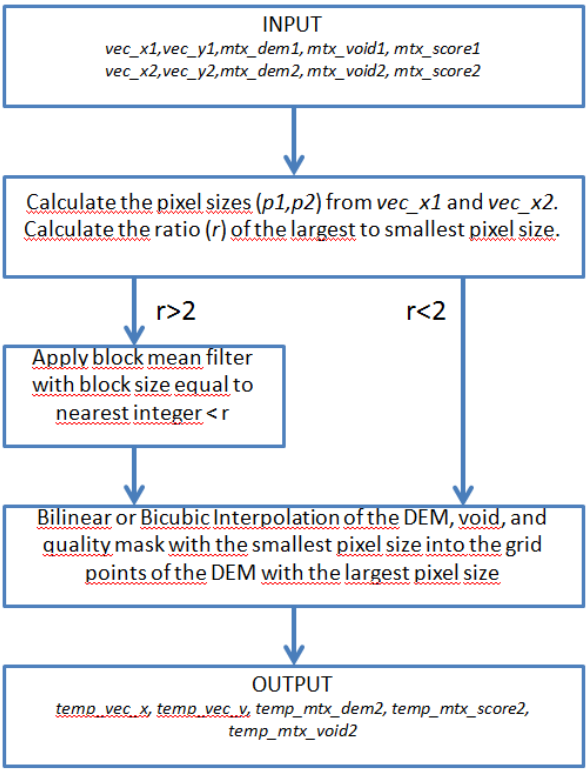

Fig. 4.2: Logical flow diagram of the resample module.

Pseudo Code

1. *Calculate pixel sizes for each of the DEMs, p1 and p2*
2. *Calculate the ratio of the largest pixel size to the smallest pixel size.*
3. *If the ratio is greater than 2, r > 2, then perform block filtering (mean) of the smallest DEM. The size of the block is rounded to the nearest integer.*
4. *Perform bilinear or bicubic interpolation of the DEM with the smaller pixel size at the grid points of the larger DEM. Resample (interpolate) also the void and score masks if available.*

4.2.2 Subtraction module

General description and definition of the objects to be computed

Performs pixel by pixel differencing of the two DEMs. The slave DEM, *temp_mtx_dem2*, has to be subtracted from the master DEM, *mtx_dem1*. The two DEMs as generated by the resample module are subtracted. The points in time (*const_time1, const_time2*) have to be provided to calculate annual changes by dividing the module output (*mtx_diff*) by the temporal difference (*const_time1-const_time2).*

Definition of variables

Input variables:

- *mtx_dem1, temp_mtx_dem2*
- *const_time1, const_time2*

Internal variables:

- *temp_vec_x*
- *temp_vec_y*

Output variables:

- *const_time1, const_time2*
- *mtx_diff*

Model equations and logical flow diagram

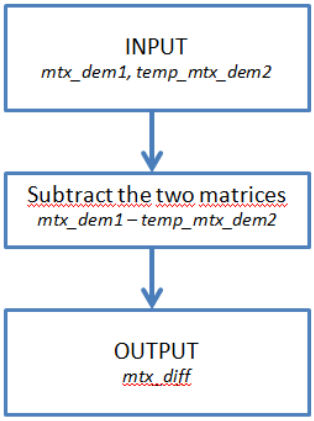

Fig. 4.3: Logical flow diagram of the subtraction module.

Pseudo Code

1. *Subtract the older DEM from the more recent one (mtx_dem1 – temp_mtx_dem2)*
2. *Optional: divide the result mtx_diff by (const_time1-const_time2*

4.2.3 Mask generator module

General description and definition of the objects to be computed

This module creates a glacier mask for later quality assessment that is consistent with the pixel size and grid of the input DEM. Specifically, it determines whether each pixel of the DEM lies within input polygons. Two separate binary masks are created where 1 is used to indicate glacier and stable terrain, respectively, to be used in the co-registration module, quality indicator module and the summary statistics module.

Definition of variables

Input variables:

- *Shapefiles from the import module (i.e. glaciers, DEM voids)*
- *vec_x1 (master)*
- *vec_y1 (master)*

Output variables:

- *mask_glac: Binary mask equal to 1 for glaciers*
- *mask_terr: Binary mask equal to 1 for stable terrain*

Model equations and logical flow diagram

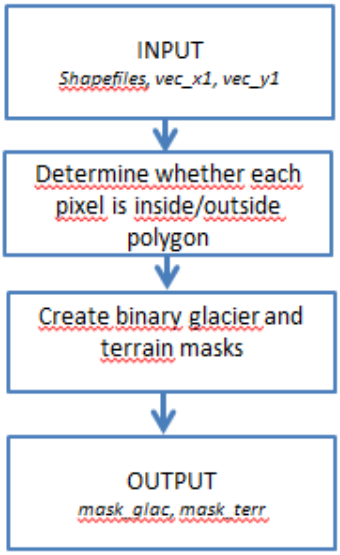

Fig. 4.4: Logical flow diagram of the mask generator module.

Pseudo Code

1. *Loop through each shapefile*
2. *Loop through each polygon object in shapefile.*
3. *Loop through each pixel coordinate in DEM.*
4. *For each pixel of one DEM matrix (master), determine whether pixel is inside or outside glacier polygon. Set mask_glac == 1 for inside, and 0 = outside to mask.For mask_terr, set 1 for outside glacier mask and 0 for inside the glacier mask*
5. *If DEM void masks are available, then similarly, loop through each individual shapefile and polygon, if inside cloud, ocean, lake or DEM void, then set both mask_glac and mask_terr = 0.*

4.2.4 Co-registration module

General description and definition of the objects to be computed

This module applies the co-registration algorithm chosen in the PVASR (Glaciers_cci, 2012a). The module uses the elevation differences and the slope and aspect grids determined from the master DEM to minimize the co-registration equation and determine the three unknown parameters using using the sample of pixels on stable terrain defined by *mask_terr*.

Definition of variables

Input variables:

- *mtx_dem1*
- *mtx_diff*
- *mask_terr*

Internal variables:

- *mtx_slp*: the slope derivative of the master DEM (*mtx_dem1*)
- *mtx_asp*: the aspect derivative of the master DEM (*mtx_dem1*)

- *mtx_difftanslope:* the difference matrix divided by the tangent of slope matrix.
- *const_param_a: amplitude of the cosine curve*
- *const_param_b: phase of the cosine curve*
- *const_param_c:mean offset of the cosine curve*
- *const_mean_slope: the mean slope of the pixels over stable terrain*

Output variables:

- *dx:* The adjustment required to add to *x_vec*
- *dy:* The adjustment required to add to *y_vec*
- *dz:* The adjustment required to add to *mtx_dem2*

Model equations and logical flow diagram

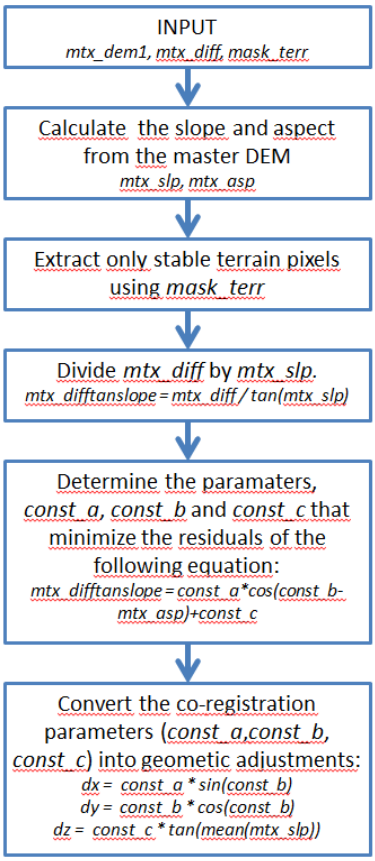

Fig. 4.5: Logical flow diagram of the co-registration module.

Pseudo Code

The pseudo code of the module is reported here.

1. *Calculate slope (mtx_slp) and aspect (mtx_asp) grids from the master DEM*
2. *Extract only stable terrain pixels of mtx_diff, mtx_slp and mtx_asp using mask_terr from the mask generator module*
3. *Divide mtx_diff by tan(mtx_slp) to produce mtx_difftanslope*
4. *Determine the three constant parameters (const_a,const_b,const_c) by optimization / minimization of the equation:*

*mtx_diftanslope = const_a*cos(const_b-mtx_asp)+const_c*

5. *Convert the determined parameters into geometric corrections:*
   *dx = const_a * sin(const_b)*
   *dy = const_b * cos(const_b)*
   *dz = const_c * tan(mean(mtx_slp))*

4.2.5 Adjustment module

General description and definition of the objects to be computed

This module adds the horizontal co-registration adjustments to the horizontal vectors and the DEM is adjusted by the vertical co-registration parameter. The output is the co-registered slave DEM.

Definition of variables

Input variables:

- *vec_x2, vec_y2*
- *mtx_dem2*
- *dx, dy, dz*

Output variables:

- *vec_x2_adj*: the co-registered x coordinate vector for the slave DEM
- *vec_y2_adj*: the co-registered y coordinate vector for the slave DEM
- *mtx_dem2_adj*: the slave DEM adjusted for mean offsets

Model equations and logical flow diagram

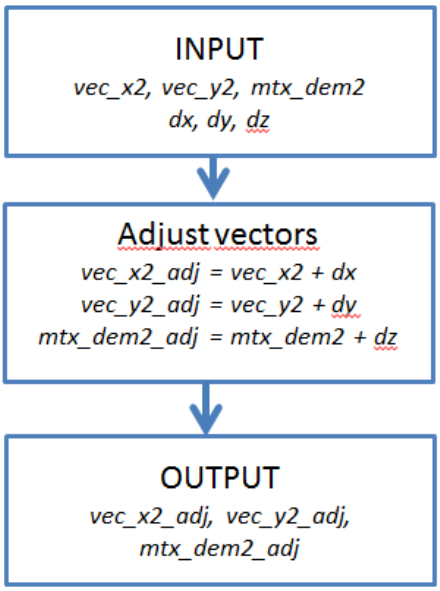

Fig. 4.6: Logical flow diagram of the adjustment module.

Pseudo Code

1. *vec_x2_adj = vec_x2 + dx*

2. *vec_y2_adj = vec_y2 + dy*
3. *mtx_dem2_adj = mtx_dem2 + dz*

4.2.6 Quality indication module

General description and definition of the objects to be computed

This module implements a number of post processing routines to derive a quality indicator mask that is output with the elevation difference grid. This includes a combination of the score matrices (if available) and the void matrices (if available).

Definition of variables

Input variables:

- *temp_vec_x, temp_vec_y*
- *mtx_dem1, temp_mtx_dem2*
- *mtx_score1, mtx_void1, temp_ mtx_score2, temp_mtx_void2*
- *const_time1, const_time2*
- *mtx_diff*
- *mask_glac*
- *mask_terr*

Output variables:

- *mtx_qual_ind1: score channel from dem 1 (if available)*
- *mtx_qual_ind2: score channel from dem 2 (if available)*
- *mtx_qual_ind3: combined void mask from both DEMs (if available) or from only one DEM if available. Otherwise, not available.*

Model equations and logical flow diagram

For the present processing model, this module checks whether the input DEMs contain quality and void grids. It is not necessary that all input DEMs contain this information, and thus, this module does not contain any pseudo code to merge all of the individual quality masks. Rather, each quality mask (if available) is exported with the elevation change grid. For void masks (if available for both DEMs) the flow diagram is:

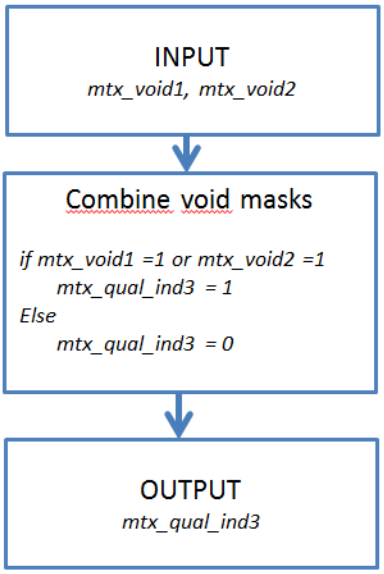

Fig. 4.7: Logical flow diagram of the quality indication module.

4.2.7 Data stack module

General description and definition of the objects to be computed
This module forms the basis for the output. It merges the elevation difference matrices and the quality indicator matrices, indicator masks (glaciers and stable terrain), etc. into one coherent data stack (data cube, multi-layer grid etc.). Since all the different layers will be in a consistent grid, they can be merged into one file. To produce a geotif, the georeference information is updated from that available of the input geotif DEM files.

Definition of variables

Input variables:

- *temp_vec_x, temp_vec_y*
- *const_time1, const_time2*
- *mtx_diff*
- *mtx_qual_ind1*:
- *mtx_qual_ind2*:
- *mtx_qual_ind**.:

Output variables:

- *one file*

Model equations and logical flow diagram
Create one multilayer file from all the individual files by merging all matrices into one 3-D matrix. This requires that each matrix has the same dimensions defined by temp_vec_x and temp_vec_y. An export function for geotif is required within the software or scripting language used.

4.2.8 Summary statistical tool module

General description and definition of the objects to be computed

This module provides the necessary statistics from the data stack over stable terrain for error budget estimates and possibly for each glacier polygon provided. In particular, it will calculate mean, median and standard deviation of the stable terrain and for glaciers. These statistics are used only as preliminary quality estimates. Users will have access to the full datasets in the datastack to make their own error assessments using their own favourite method.

Definition of variables

Input variables:

- *Geotiff from datastack*

Output variables:

- Statistics in the form of a data table to go into the header
- Name of the input datasets
- Time of the input datasets

Model equations and logical flow diagram

Mean, median, standard deviation, and confidence intervals are computed for the stable terrain and for glaciers. The statistics do not consider the spatial autocorrelation of the difference datasets.

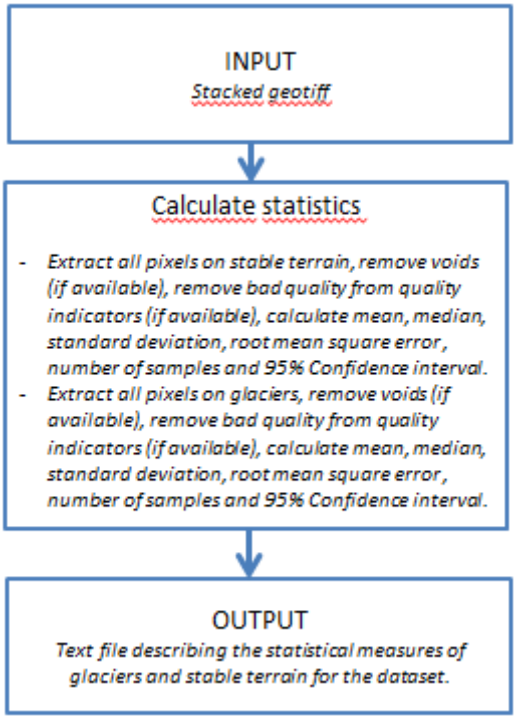

Fig. 4.7: Logical flow diagram of the statistical module.

5. Elevation change from altimetry

5.1 General macro-module description

The Repeat-Altimetry macro-module takes as input the ICESat laser acquisitions, the DEM, a down scale factor for creating a grid space and other parameters needed for the density and spatial filtering of the data and for the determination of the confidence interval of the estimated elevation trend. It evaluates:

- the coordinates of the centre of each grid cell;
- the temporal average over the time seasons of the elevation resulting from the average of the laser measurements whose locations are inside each grid cell (spatial average);
- the elevation change and the error.

The macro-module uses a support module for checking the coordinate reference systems used by the two main datasets, i.e. the ICESat data and the DEM elevation. In case it is necessary, the coordinates are transformed to a common reference system.

In the following description, the ICESat location coordinates are already converted to the DEM coordinate system, which is usually expressed in a UTM projection with datum WGS-84. It is important to note that the UTM coordinate reference system is the DEM reference system afterwards. However, the DEM can also be converted to any coordinate system, but must be consistent with the coordinate system used for the ICESat data. The choice of converting the ICESat measurement locations into the DEM coordinate system does not affect the implementation of the algorithm, but it avoids a manual input from the analyst on the common reference system to use and consequently avoids converting both the input datasets. Therefore, from a system engineers perspective, it is important to define a system conversion of projections that is flexible and that can be used with different DEMs. In fact, the new location coordinates are thought to be saved in two extra columns (numbered as 15 and 16 in Fig. 5.1) to add to the matrix containing the original ICESat data as they are extracted from the NASA archive. Actually, the ICESat level 1B GLAS06 data that are extracted are delivered in granules, which are product-specific data packets distributed as scaled integer binary files containing fixed-length records. Thus, the input interface showed in Fig. 5.1 allows reading and converting them as an ASCII file, with a specific format, described in detail in the IODD, by using the NGAT software available to download at the ICESat NSIDC project website (http://nsidc.org/data/icesat/tools.html). This software runs properly under the IDL (Interactive Data Language) environment or by using the IDL Virtual Machine. Figure 5.1 also shows the inner structure of the macro-module. It includes the following modules:

- Altimetry data pre-filtering module;
- DEM interpolation module;
- Grid-space module;
- DEM subtraction module;
- Time series module;
- Density and spatial filtering module;
- Altimetry elevation, trend and error analysis module;
- DEM mask and output format module.

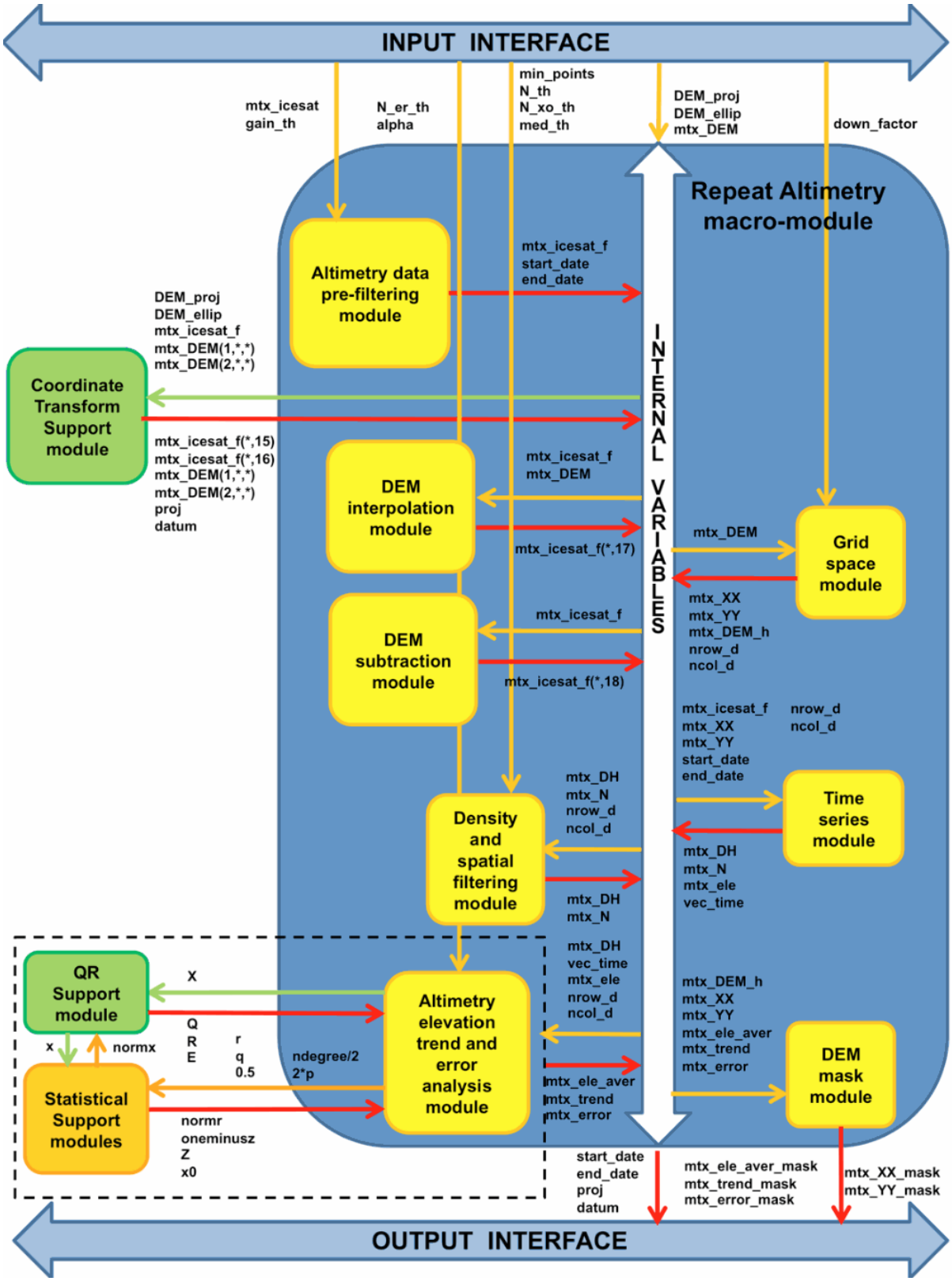

Fig. 5.1: Structure of the Repeat Altimetry macro-module.

The coordinate transform module, the QR module and the statistical modules constitute the support modules called by the macro-module. More details on the modules included in the Statistical Support modules block are shown in Fig. 5.2. The time sequence is given by the access of each module to the internal memory and from the consequent variable flow. The temporal order in Fig. 5.1 goes from top to bottom.

As mentioned above, the input interface prepares the ICESat and the DEM input data in the format required to allow correct functioning of the repeat altimetry macro-module. On the other end, the output interface makes the output data of the processing system available in the format described in the PSD (Glaciers_cci, 2011a). Details about the individual processing steps in each module are provided in the respective sections below.

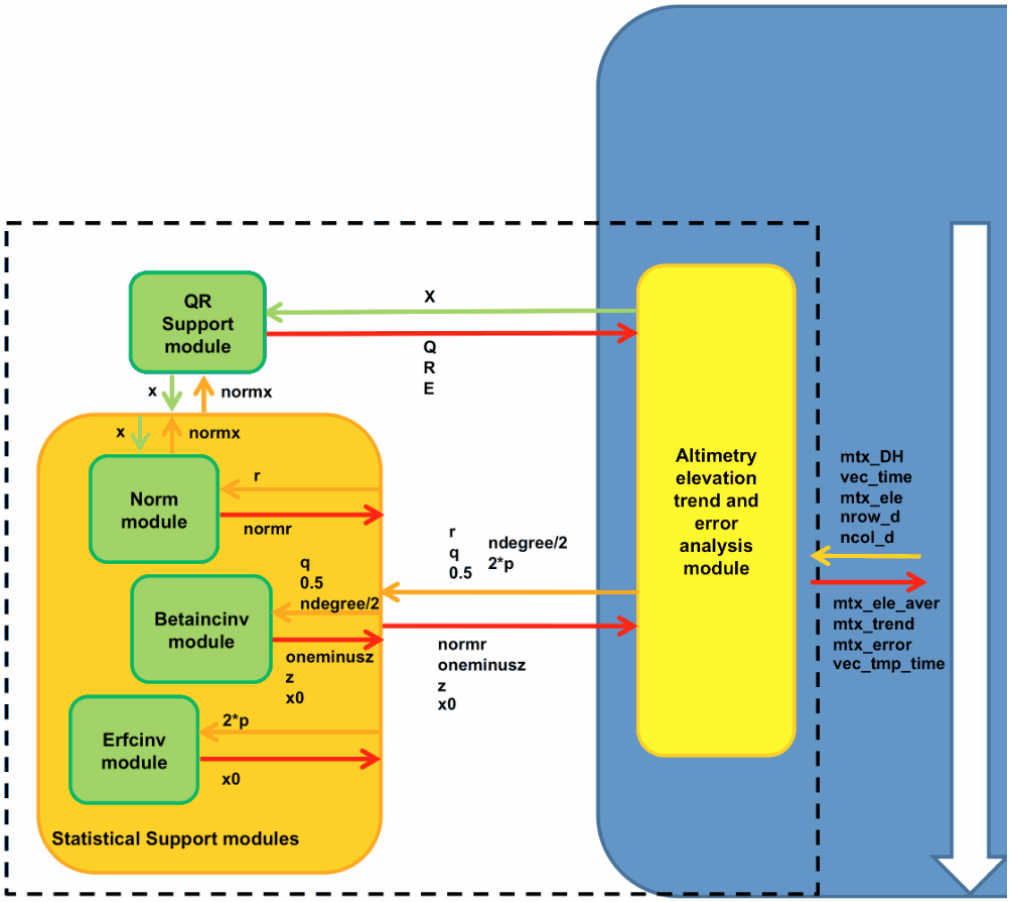

Fig. 5.2: Details on the Statistical Support modules block and its interaction with the Repeat Altimetry macro-module.

5.2 Modules description

5.2.1 Altimetry data pre-filtering module

General description
This module filters out ICESat measurements not suitable for the estimation of elevation changes.

Definition and detailed description of the objects to be computed
Records with gain correction values of -999 are deleted and data point measurements with receiver gain values larger than a fixed threshold (to be decided by the analyst) are discarded.

Definition of variables

Input variables:

- *mtx_icesat* = identifies the ICESat data (2D array of real elements, for the meaning of each column of the array please refer to the IODDv1);
- *gain_th* = gain threshold (scalar integer, non-dimensional).

Internal variables:

- *gain_ind* = identifies the index of the rows and consequently the whole elements of the 2D array *mtx_icesat* to keep (1D array of double elements, non-dimensional);
- *m_m* = identifies the minimum and maximum year of acquisition, the minimum and maximum month of acquisition and the minimum and maximum day of acquisition (scalar integer);
- *date_ind* = 1D array containing the indices of the elements which will be extracted from the *mtx_icesat* 2D array for the determination of the starting and ending date of acquisition (1D array of real elements, non-dimensional);
- *mtx_date* = 2D array with two columns representing the month and the day of acquisition, with the order here stated (2D array of integer elements, non-dimensional);
- *vec_date* = 1D array representing the day of acquisition (1D array of integer elements, non-dimensional).

Output variables:

- *mtx_icesat_f* = filtered ICESat data (2D array of real elements having the same format of the *mtx_icesat* 2D array);
- *start_date* = start of acquisition (string of 8 chars);
- *end_date* = end of acquisition (string of 8 chars).

Model equations and logical flow diagram
This module performs a query over the dataset given by the ICESat measurements and consequently no analytical equations are implemented. Figure 5.3 shows the logical flow diagram.

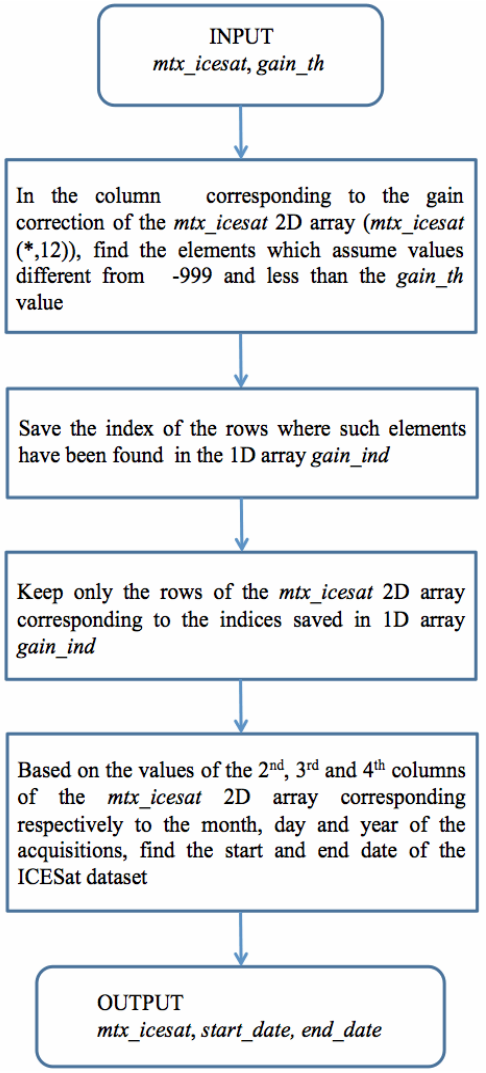

Fig. 5.3: Logical flow diagram of the altimetry data pre-filtering module.

Pseudo Code

1. *Find where mtx_icesat(*,12) is not equal to -999 and where mtx_icesat(*,12) is less than the gain threshold, gain_th*
2. *Save the result of such query in the gain_ind 1D array*
3. *Save in the mtx_icesat_f 2D array the rows of the mtx_icesat 2D array corresponding to the indices recorded in the gain_ind array*
4. *Find the minimum value of the elements of the column representing the year of acquisition (mtx_icesat_f(*,4))*
5. *Assign this value to the variable m_m*
6. *Transform this numerical value in a string and save it into the string variable start_date*
7. *Find where mtx_icesat_f(*,4) is equal to m_m*
8. *Save the result of such query in the date_ind 1D array*
9. *Extract the elements of the mtx_icesat whose row indices correspond to the indices saved in the data_ind 1D array and whose columns are the 2nd and the 3rd.*
10. *Save the extracted elements into the 2D array, mtx_date*

11. *Find the minimum value of the elements of the first column of the 2D array, mtx_date, representing the month of acquisition (mtx_date(*,1))*
12. *Assign this value to the variable m_m*
13. *Transform this numerical value in a string and concatenate it to the string variable start_date*
14. *Find where mtx_date(*,1) is equal to m_m*
15. *Save the result of such query in the date_ind 1D array*
16. *Extract the elements of the 2nd column of the mtx_date 2D array whose row indices correspond to the indices saved in the data_ind 1D*
17. *Save the extracted elements into the 1D array, vec_date*
18. *Find the minimum element of the vec_date array*
19. *Assign this value to the variable m_m*
20. *Transform this numerical value in a string and concatenate it to the string variable start_date*
21. *Find the maximum value of the elements of the column representing the year of acquisition (mtx_icesat_f(*,4))*
22. *Assign this value to the variable m_m*
23. *Transform this numerical value in a string and save it into the string variable end_date*
24. *Find where mtx_icesat_f(*,4) is equal to m_m*
25. *Save the result of such query in the date_ind 1D array*
26. *Extract the elements of the mtx_icesat whose row indices correspond to the indices saved in the data_ind 1D array and whose columns are the 2nd and the 3rd.*
27. *Save the extracted elements into the 2D array, mtx_date*
28. *Find the maximum value of the elements of the first column of the 2D array, mtx_date, representing the month of acquisition (mtx_date(*,1))*
29. *Assign this value to the variable m_m*
30. *Transform this numerical value in a string and concatenate it to the string variable end_date*
31. *Find where mtx_date(*,1) is equal to m_m*
32. *Save the result of such query in the date_ind 1D array*
33. *Extract the elements of the 2nd column of the mtx_date 2D array whose row indices correspond to the indices saved in the data_ind 1D*
34. *Save the extracted elements into the 1D array, vec_date*
35. *Find the maximum element of the vec_date array*
36. *Assign this value to the variable m_m*
37. *Transform this numerical value in a string and concatenate it to the string variable start_date*

5.2.2 DEM interpolation module

General description

This module interpolates the elevations of the DEM at the locations identified by the laser footprint.

Definition and detailed description of the objects to be computed

The object to compute in this module is the elevation, h_i', at the location of each laser acquisition by using surrounding elevation values available from the DEM. The DEM elevations to consider are those included in a rectangular area around each location of acquisition whose extension is determined by the DEM resolution.

Definition of variables

Input variables:

- *mtx_icesat_f* = identifies the filtered ICESat data (2D array of real elements having the same format of the *mtx_icesat* 2D array);
- *mtx_DEM* = DEM (3D array of real elements, for the format of the array please refer to the IODDv1).

Internal variables:

- *demx* = identifies the horizontal DEM resolution (along the x-axis) (scalar real, m);
- *demy* = identifies the vertical DEM resolution (along the y-axis) (scalar real, m);
- *i* = counter (scalar real, non-dimensional);
- *ind_X* = 1D array containing the horizontal indices (along the x-axis) of the DEM elements whose distance from the horizontal coordinate of the laser acquisition is lower or equal to the horizontal DEM resolution (1D array of real elements, non-dimensional);
- *ind_Y* = 1D array containing the vertical indices (along the y-axis) of the DEM elements whose distance from the vertical coordinate of the laser acquisition is lower or equal to the vertical DEM resolution (1D array of real elements, non-dimensional);
- *N* = number of DEM elevations found around each laser footprint location (scalar integer, non-dimensional);
- *x* = array containing the horizontal coordinates of the DEM identified points (initially it is a 2D array of real elements and later a 1D array of real elements, m);
- *y* = array containing the vertical coordinates of the DEM identified points (initially it is a 2D array of real elements and later a 1D array of real elements, m);
- *z* = array containing the elevations of the DEM identified points (initially it is a 2D array of real elements and later a 1D array of real elements, m);
- *xyz* = 2D array of *N*x3 elements (2D array of real elements, m);
- *sxyz* = sorted 2D array of *N*x3 elements (2D array of real elements, m);
- *xy* = 1D array of *N* complex elements; the real part of each of them corresponds to the horizontal coordinate, whereas the imaginary part corresponds to the vertical coordinate of the DEM identified points (1D array of complex elements);
- *d* = array representing either the distance between each possible couple of DEM points (2D array of *N*x*N* real elements, m) or the distance between the laser footprint location and each point of the DEM (1D array of 1x*N* real elements, m);

- *g* = array representing the Green function of the biharmonic operator used in the interpolation method (it is a 2D array of *N*x*N* real elements for the determination of the weights and a 1D array of 1x*N* real elements for the computation of the interpolated elevation, see next section);
- *w* = 1D array representing the weights of the interpolation method (see next section) (1D array of *N*x1 real elements);
- *hi* = interpolated result (scalar real, m);
- *zi* = complex number, the real part is the horizontal coordinate of the laser footprint location and the imaginary part is the vertical coordinate of the laser footprint location (scalar complex);
- *mask* = 1D array of the indices where the distance is zero (1D array of integer elements, non-dimensional).

Output variables:

- *mtx_icesat_f* = modified filtered ICESat data; the format is the same of the *mtx_icesat* with the addition of a further column containing the interpolated results obtained at each laser footprint location (2D array of real elements).

Model equations and logical flow diagram

This module is mainly based on the equations reported in Sandwell (1987) where a biharmonic spline interpolation method has been developed mainly for the interpolation of irregular spaced satellite altimetry profiles. The theory is here explained using the same nomenclature defined in the list of variables reported in the previous section. For simplicity, it is better to represent the *k-th* laser measurement, at the acquisition time $t = t_k^m$, as a point in a 3D space:

$$P_k^m = \left(x_k^m, y_k^m, h_k^m\right) \tag{5.1}$$

where x_k^m, y_k^m represent the location of the acquisition (i.e. the latitude and longitude, which for simplicity are assumed expressed in the same reference system of the DEM), h_k^m, represents the elevation above the reference ellipsoid (WGS84) and the upper index *m* stands for measurements. In terms of variables:

- x_k^m corresponds to the element (*k*,16) of the 2D array *mtx_icesat_f*;
- y_k^m corresponds to the element (*k*,15) of the 2D array *mtx_icesat_f*;
- h_k^m corresponds to the element (*k*,10) of the 2D array *mtx_icesat_f*.

Suppose that *N* DEM points are present around the location footprint identified by x_k^m, y_k^m. The problem is to find a biharmonic 2D function that passes through these data points and then compute the value that this function assumes at x_k^m, y_k^m. This value has been indicated in the variables list as *hi*.
In a 2D dimensional space, the problem is formulated as follows:

$$\begin{cases} \nabla^4 f(\mathbf{x}) = \sum_{j=1}^{N} w_j \, \delta(\mathbf{x} - \mathbf{x}_j) \\ f(\mathbf{x}_i) = f_i \end{cases} \tag{5.2}$$

where ∇^4 is the biharmonic operator, $\mathbf{x}$ is a position in the 2D space (i.e., (x, y)). The general solution is a linear combination of Green functions centred at each data point:

$$f(\mathbf{x}) = \sum_{j=1}^{N} w_j \, g(\mathbf{x} - \mathbf{x}_j) \tag{5.3}$$

where g is the 2D space Green function given by the following equation:

$$g(\mathbf{x}) = |\mathbf{x}|^2 \ln(|\mathbf{x}| - 1) \tag{5.4}$$

The w_j's are found by solving the linear system:

$$f(\mathbf{x}_i) = f_i = \sum_{j=1}^{N} w_j \, g(\mathbf{x}_i - \mathbf{x}_j) \qquad i = 1 \ldots N \tag{5.5}$$

which, using a matrix representation, assumes the following expression:

$$\begin{bmatrix} g(\mathbf{x}_1 - \mathbf{x}_1) & g(\mathbf{x}_1 - \mathbf{x}_2) & \ldots & g(\mathbf{x}_1 - \mathbf{x}_N) \\ g(\mathbf{x}_2 - \mathbf{x}_1) & g(\mathbf{x}_2 - \mathbf{x}_2) & \ldots & g(\mathbf{x}_2 - \mathbf{x}_N) \\ \vdots & \vdots & \vdots & \vdots \\ g(\mathbf{x}_N - \mathbf{x}_1) & g(\mathbf{x}_N - \mathbf{x}_2) & \ldots & g(\mathbf{x}_N - \mathbf{x}_N) \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_N \end{bmatrix} \tag{5.6}$$

The generic difference $\mathbf{x}_i - \mathbf{x}_j$ represents the distance between the *i-th* and the *j*-th data points, i.e. the distance between each possible couple of the *N* DEM points identified in the proximity of the laser footprint location. In the list of the variables, this distance has been indicated as *d*. The f_i values correspond to the DEM elevation values.

In a compact form, the equation (5.6) becomes:

$$\mathbf{G\,W = F} \tag{5.7}$$

Thus, for the weights determination it follows:

$$\mathbf{W = G^{-1} F} \tag{5.8}$$

The interpolated elevation, *hi*, at the footprint location x_k^m, y_k^m is given by:

$$hi = f(\mathbf{x}_k^m) = \sum_{j=1}^{N} w_j \, g(\mathbf{x}_k^m - \mathbf{x}_j) \tag{5.9}$$

which, using a matrix representation, assumes the following expression:

$$\left[g\left(\mathbf{x}_k^m-\mathbf{x}_1\right)\quad g\left(\mathbf{x}_k^m-\mathbf{x}_2\right)\quad\ldots\quad g\left(\mathbf{x}_k^m-\mathbf{x}_N\right)\right]\begin{bmatrix}w_1\\w_2\\\vdots\\w_N\end{bmatrix}=hi \tag{5.10}$$

The generic difference $\mathbf{x}_k^m-\mathbf{x}_j$ represents the distance between the point that identifies the laser footprint and the *j-th* DEM data points. In the list of the variables, this distance has been indicated as *d*. Figure 5.4 shows the logical flow diagram.

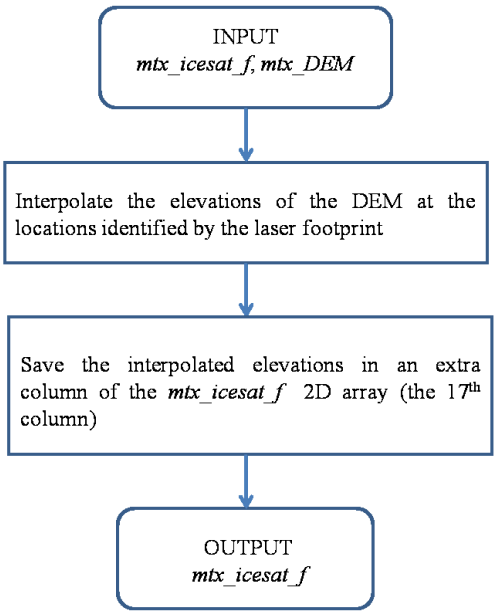

Fig. 5.4: Logical flow diagram of the DEM interpolation module.

Pseudo Code

1. *Compute the difference mtx_DEM(1,1,2)-mtx_DEM(1,1,1)*
2. *Assign the result to the variable demx*
3. *Compute the difference mtx_DEM(2,2,1)-mtx_DEM(2,1,1)*
4. *Assign the result to the variable demy*
5. *Initialise the counter i to 1*
6. *Repeat the following instructions until the counter i is lower or equal to the number of rows of the mtx_icesat_f:*
    - *find the indices of the elements in the mtx_DEM array where the absolute value of the difference between mtx_DEM(1,1,*) and mtx_icesat_f(i,16) is lower or equal to demx*
    - *save the results in the 1D array ind_X*

- *find the indices of the elements in the mtx_DEM array where the absolute value of the difference between mtx_DEM(2,*,1) and mtx_icesat_f(i,15) is lower or equal to demy*
- *save the results in the 1D array ind_Y*
- *if the 1D array ind_X is not empty and if the 1D array ind_Y is not empty then:*
  - *if the length of ind_X is greater than 1 and if the length of ind_Y is greater than 1 then:*
    - *assign the content of mtx_DEM(1,ind_Y,ind_X) to the variable x*
    - *assign the content of mtx_DEM(2,ind_Y,ind_X) to the variable y*
    - *assign the content of mtx_DEM(3,ind_Y,ind_X) to the variable z*
    - *find the number of elements in the 2D array x*
    - *assign the result to the variable N*
    - *reshape the x 2D array as a 1D column array of N elements*
    - *reshape the y 2D array as a 1D column array of N elements*
    - *reshape the z 2D array as a 1D column array of N elements*
    - *create the 2D array xyz in the way that the 1D array x corresponds to the first column, the 1D array y to the second column and the 1D array z to the third column*
    - *sort the rows of the xyz 2D array in ascending order for the second column first, and then by ascending order for the first column*
    - *assign the result of the sorting to the 2D array sxyz*
    - *assign the first column of sxyz to the 1D column array x*
    - *assign the second column of sxyz to the 1D column array y*
    - *assign the third column of sxyz to the 1D column array z*
    - *create the 1D array of Nx1 complex numbers xy; each element of the array has as a real part the corresponding element of the array x and as imaginary part the corresponding element of the array y*
    - *create the 2D array of NxN elements d; each column of the array is a replica of the xy column array*
    - *compute the absolute value of the difference between d and its array transpose*
    - *assign the result to the 2D array d*
    - *replace the zeros along the diagonal elements of d with ones*
    - *compute the square of each element of the 2D array d and multiply, element by element, this result for the result obtained applying, to each element of the 2D array d, first the natural logarithm and then the subtraction of the quantity 1*
    - *assign the result of this computation to the 2D array g*
    - *replace the element along the diagonal of g with zeros*
    - *compute the backslash or matrix left division between the 2D array g and the 1D array z (this corresponds first to compute the inverse of the 2D array g and then to compute the matrix multiplication between the result and the 1D array z)*
    - *assign the result to the 1D array w*
    - *assign zero to the variable hi*
    - *compute the array transpose of the 1D array xy*
    - *assign the result to the 1D array xy*

- *create the complex variable zi; the real part is the mtx_icesat_f(i,16) element and the imaginary part is the mtx_icesat_f(i,15) element*
- *compute the absolute value of the difference between zi and each element of the 1D row array xy*
- *assign the result to the 1D array d*
- *find the indices where the elements of the 1D array d are equal to 0*
- *assign the result of this query to the 1D array mask*
- *if mask is not an empty array then replace the zeros in the 1D array with ones*
- *compute the square of each element of the 1D array d and multiply, element by element, this result for the result obtained applying, to each element of the 1D array d, first the natural logarithm and then the subtraction of the quantity 1*
- *assign the result of this computation to the 1D array g*
- *if mask is not an empty array then replace the elements of the 1D array g, at the positions correspondent to the elements of the mask array, with zeros*
- *compute the matrix multiplication between the 1D array g and the 1D array w*
- *assign the result of this computation to the variable hi*

- *else compute the average value of the elements in mtx_DEM(3,ind_Y,ind_X) and assign the result to hi*

- *else assign NaN to hi*
- *save the variable hi in an extra column of the 2D array mtx_icesat_f, mtx_icesat_f(i,17)*
- *increment the counter i of 1*

5.2.3 Grid-space module

General description
This module creates a 2-dimensional grid-space.

Definition and detailed description of the objects to be computed
For creating three-dimensional representations of data, it is necessary to have data defined over an evenly spaced, two-dimensional grid. The one created in this module is based on the available DEM, which is simply down-sampled by using the value of the down factor provided as input though the *down_factor* variable. The module also provides the information contained in the DEM down-sampled over the grid space created.

Definition of variables

Input variables:

- *mtx_DEM* = DEM (3D array of real elements, see IODDv1 for format).

Output variables:

- *mtx_XX* = identifies the result of the down-sampling applied to the 2D array corresponding to *mtx_DEM(1,*,*)* (2D array of real elements, m);
- *mtx_YY* = identifies the result of the down-sampling applied to the 2D array corresponding to *mtx_DEM(2,*,*)* (2D array of real elements, m);
- *mtx_DEM_h* = identifies the result of the down-sampling applied to the 2D array corresponding to *mtx_DEM(3,*,*)* (2D array of real elements, m);
- *nrow_d* = identifies the number of columns of the grid (scalar long integer, non-dimensional);
- *ncol_d* = identifies the number of rows of the grid (scalar long integer, non-dimensional).

Model equations and logical flow diagram

This module performs a down sampling of the 2D arrays stacked in the 3D *mtx_DEM* and consequently no analytical equations are implemented.

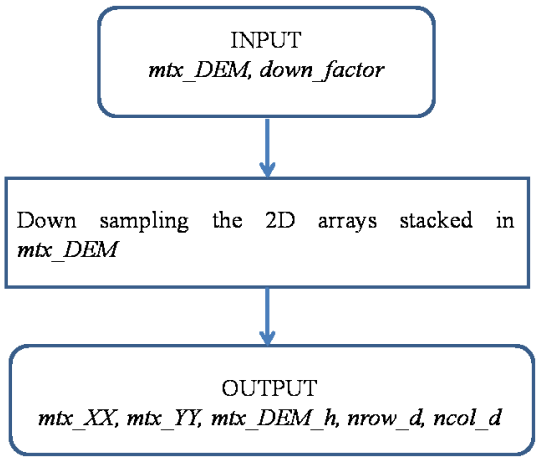

Fig. 5.5: Logical flow diagram of the Grid space module.

Pseudo Code

1. *Starting from the element (1,1) of the 2D array corresponding to mtx_DEM(1,*,*), extract every n-th element out of the array both per row and per column; the value of n corresponds to the value assumed by the variable down_factor*
2. *Save the sampled elements in the 2D array mtx_XX*
3. *Starting from the element (1,1) of the 2D array corresponding to mtx_DEM(2,*,*), extract every n-th element out of the array both per row and per column; the value of n corresponds to the value assumed by the variable down_factor*
4. *Save the sampled elements in the 2D array mtx_YY*
5. *Starting from the element (1,1) of the 2D array corresponding to mtx_DEM(3,*,*), extract every n-th element out of the array both per row and per column; the value of n corresponds to the value assumed by the variable down_factor*
6. *Save the sampled elements in the 2D array mtx_DEM_h*
7. *Find the number of rows of the 2D array mtx_XX*

8. *Assign the result to the variable nrow_d*
9. *Find the number of columns of the 2D array mtx_XX*
10. *Assign the result to the variable ncol_d*

5.2.4 DEM subtraction module

General description
This module calculates the differences between the laser measured elevations and the DEM interpolated elevations.

Definition and detailed description of the objects to be computed
The object to compute in this module is the simple difference $h_i^m - h_i'$ relative to each location of acquisition, i.e. $\left(x_i^m, y_i^m\right)$. It is important to note that the saturation elevation correction needs to be added to the elevations measured.

Definition of variables

Input variables:

- *mtx_icesat_f* = identifies the modified filtered ICESat data (2D array of real elements having the same format of the *mtx_icesat_f* 2D array where the column representing the DEM interpolated elevations has been added).

Output variables:

- *mtx_icesat_f* = modified filtered ICESat data; it has the same format of the *mtx_icesat_f* provided as input with the addition of a further column containing the differences between the laser measured elevations and the DEM interpolated elevations (2D array of real).

Model equations and logical flow diagram
Given the *i-th* laser measurement at the acquisition time $t = t_i^m$ as a point in a 3D space:

$$P_i^m = \left(x_i^m, y_i^m, h_i^m\right) \tag{5.11}$$

and using the same notation for indicating the DEM interpolated value at the same location $\left(x_i^m, y_i^m\right)$, but referring to the reference time $t = t_{REF}$:

$$P_i' = \left(x_i^m, y_i^m, h_i'\right) \tag{5.12}$$

the difference between GLAS measured elevations and the DEM interpolated elevations is given by:

$$\Delta h_{t_i^m - t_{REF}}\left(x_i^m, y_i^m\right) = h_i^m - h_i' + \Delta h_i \tag{5.13}$$

The third term in the above equation represents the saturation elevation correction. In terms of variables it is:

- h_i^m corresponds to the element (i,10) of the 2D array *mtx_icesat_f*;
- h_i'corresponds to the element (i,17) of the 2D array *mtx_icesat_f*;
- Δh_i corresponds to the element (i,12) of the 2D array *mtx_icesat_f*.

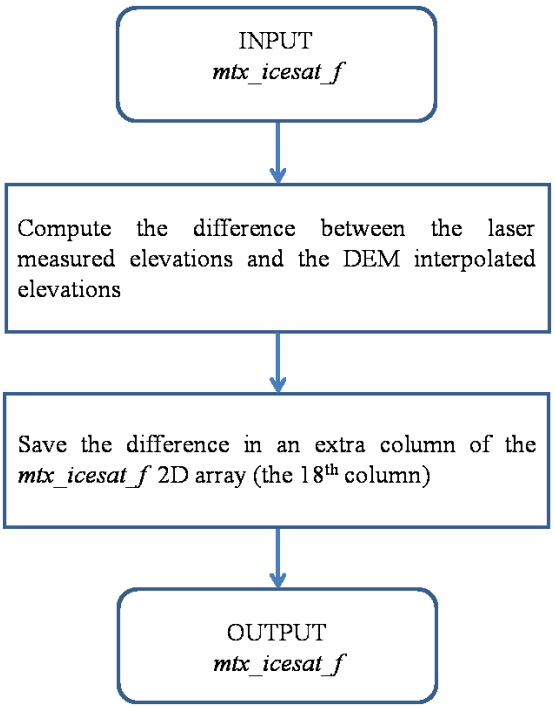

Fig. 5.6: Logical flow diagram of the DEM subtraction module.

Pseudo Code

1. *Add element by element the 10th column of the mtx_icesat_f 2D array to the 12th column of the same 2D array and then subtract, element by element, the 17th column of the same 2D array*
2. *Save the result in an extra column of the 2D array mtx_icesat_f, mtx_icesat_f(*,18)*

5.2.5 Time series module

General description

This module creates a time series considering 3 seasons for each year of acquisition. Thus, each time step, or epoch, is represented by a single season. The first season corresponds to the period from the 1st January to the 30th of April; the second corresponds to the period from the 1st May to the 31st of August and the third to the period from the 1st of September to the 31st of December.

Definition and detailed description of the objects to be computed

The objects to compute in this module are essentially 3D arrays for representing the measured elevations and the difference between the latter and the DEM interpolated elevations in a grid

format. The average is computed considering the number of measurements and the number of difference points in each grid cell, respectively. This information is also stored in 3D arrays which can be used for statistical analysis. A 3D array type variable has been chosen because each time step can be represented by the value that the first index of the array assumes. The grid space is based on the *mtx_XX* and *mtx_YY* arrays, which identify the result of the down-sampling applied to the 2D array corresponding to *mtx_DEM(1,*,*)* and to *mtx_DEM(2,*,*)*, respectively. This module needs also to compute the mean time for each time step, expressed in years.

Definition of variables

Input variables:

- *mtx_icesat_f* = identifies the filtered ICESat data (2D array of real elements having the same format as the *mtx_icesat* 2D array);
- *mtx_XX* = identifies the result of the down-sampling applied to the 2D array corresponding to *mtx_DEM(1,*,*)* (2D array of real elements, m);
- *mtx_YY* = identifies the result of the down-sampling applied to the 2D array corresponding to *mtx_DEM(2,*,*)* (2D array of real elements, m);
- *start_date* = start of acquisition (string of 8 chars);
- *end_date* = end of acquisition (string of 8 chars);
- *nrow_d* = identifies the number of columns of the grid (scalar long integer, non-dimensional);
- *ncol_d* = identifies the number of rows of the grid (scalar long integer, non-dimensional).

Internal variables:

- *int_index* = time step counter (scalar integer, non-dimensional);
- *int_year* = identifies the counter representing the years of acquisition (scalar integer, non-dimensional);
- *ys* = first year of acquisition (scalar integer, non-dimensional);
- *ye* = last year of acquisition (scalar integer, non-dimensional);
- *int_seas* = identifies the counter for the three seasons (scalar integer, non-dimensional);
- *months* = 1D array identifying the months of each season (1D array of integer elements, non-dimensional);
- *days* = 1D array identifying the days of each month (2D array of integer elements, non-dimensional);
- *vec_X* = 1D array containing the horizontal coordinates (along the x-axis) of the laser footprint location (1D array of real elements, m);
- *vec_Y* = 1D array containing the vertical coordinates (along the x-axis) of the laser footprint location (1D array of real elements, m);
- *vec_diff* = 1D array containing the difference between the laser measured elevations and the DEM interpolated elevations (1D array of real elements, m);
- *vec_ele* = 1D array containing the laser measurements (1D array of real elements, m);

- *m_diff* = mean value of the difference between the laser measured elevations and the DEM interpolated elevations whose locations are inside a given grid cell (scalar real, m);
- *m_ele* = mean value of the laser measured elevations whose locations are inside a given grid cell (scalar real, m);
- *N* = number of elevation measurements whose locations are inside a given grid cell (scalar integer, non-dimensional);
- *dx* = identifies the horizontal grid resolution (along the *x*-axis) (scalar real, m);
- *dy* = identifies the vertical grid resolution (along the *y*-axis) (scalar real, m);
- *i* = counter (scalar long integer, non-dimensional);
- *j* = counter (scalar long integer, non-dimensional);
- *centreX* = identifies the *x*-coordinates of the centre of each grid cell (scalar real, m);
- *centreY* = identifies the *y*-coordinates of the centre of each grid cell (scalar real, m);
- *ind_row* = 1D array identifying the indices of the rows of the 2D array *mtx_icesat_f* whose relative elevation measurements are acquired at a specific time within a given season (1D array of real elements, non-dimensional);
- *vec_ind* = 1D array identifying the indices of the horizontal coordinates (along *x*-axis) and the vertical coordinates (along *y*-axis) of the elevation measurements whose locations are inside a given grid cell (1D array of real elements, non-dimensional).

Output variables:

- *mtx_DH* = 3D array where each time step is placed into different layer, corresponding to the first index of the array; each layer then represents the averaging results of the difference between the laser measured elevations and the DEM interpolated elevations; the average is applied for representing such quantity in a grid format (3D array of real elements, m);
- *mtx_ele* = 3D array where each time step is placed into different layer, corresponding to the first index of the array; each layer then represents the results of averaging the laser measured elevations; the average is applied for representing the elevation in a grid format (3D array of real elements, m);
- *mtx_N* = 3D array where each time step is placed into different layer, corresponding to the first index of the array; each layer then represents the number of the data points used within each grid cell for averaging both the laser measurements and the difference between the laser measured elevations and the DEM interpolated elevations (3D array of real elements);
- *vec_time* = 1D array representing the mean time at each time step (1D array of real elements, year).

Model equations and logical flow diagram

The measured elevations are averaged in order to represent them over a grid space:

$$\bar{h}_{\bar{t}}^{m}(x_c, y_c) = \frac{\sum_{k=1}^{N} h_k^m}{N} \tag{5.14}$$

N is the number of laser acquisitions within a fixed range distance from the centre of each grid cell, (x_c, y_c). This distance has been fixed to half the grid cell resolution. In the same way, the differences achieved for each laser footprint, $\Delta h_{t_i^m - t_{REF}}(x_i^m, y_i^m) = h_i^m - h_i'$, are averaged in order to represent the variation of the elevation on the same grid space. That is:

$$\overline{\Delta h}_{\bar{t}}(x_c, y_c) = \frac{\sum_{k=1}^{N} \Delta h_{t_k^m - t_{REF}}(x_k^m, y_k^m)}{N} = \frac{\sum_{k=1}^{N} h_k^m - h_k'}{N} \tag{5.15}$$

where t_k^m is the time of the each acquisition within the time step, or epoch, considered.

The sub-index $\bar{t}$ in both the equations (5.14) and (5.15) is the mean time value associated with the averaged elevation and averaged difference $\overline{\Delta h}$ at each time step:

$$\bar{t} = \frac{\sum_{k=1}^{N} t_k^m}{N} \tag{5.16}$$

This procedure is repeated for each season. Figure 5.7 shows the logical flow diagram.

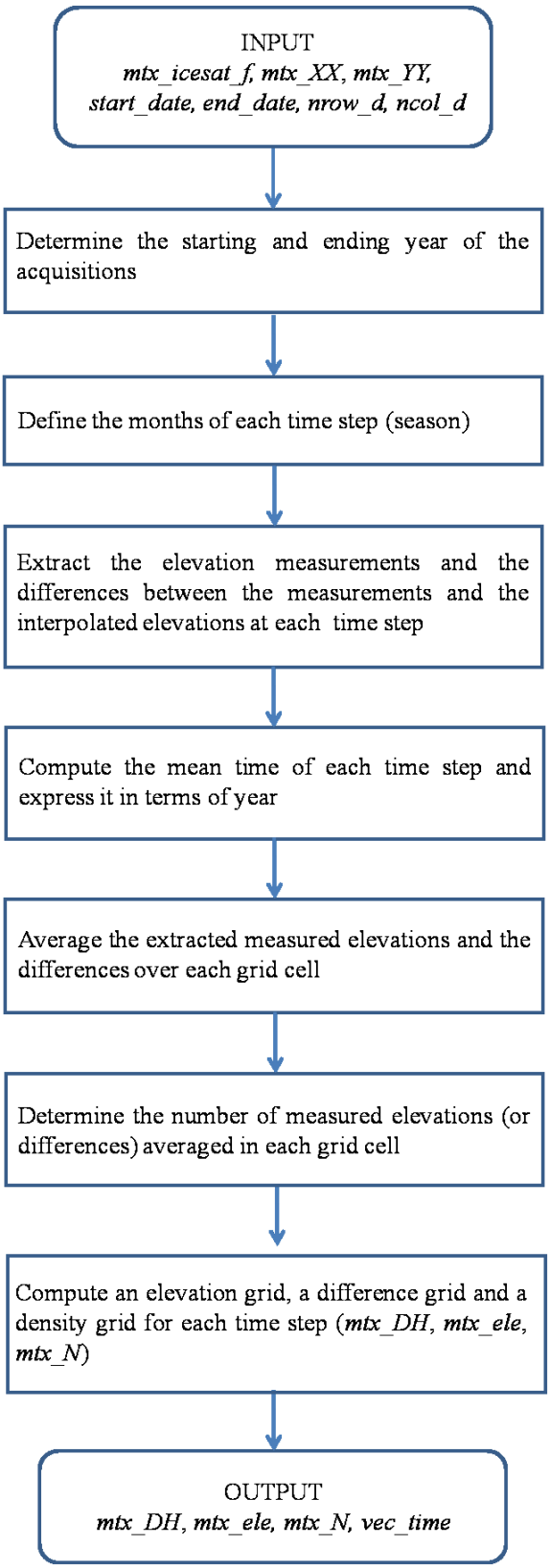

Fig. 5.7: Logialc flow diagram of the DEM subtraction module.

In the following and throughtout the rest of the document, 'NaN' mean 'Not a Number'.

Pseudo Code

1. *Compute the absolute value of the difference between mtx_XX(1,1) and mtx_XX(1,2)*
2. *Assign the result to the variable dx*
3. *Compute the absolute value of the difference between mtx_YY(1,1) and mtx_YY(2,1)*
4. *Assign the result to the variable dy*
5. *Read the first 4 characters of the string variable start_date and transform the content to an integer type*
6. *Assign this integer value to the variable ys*
7. *Read the first 4 characters of the string variable end_date and transform the content to an integer type*
8. *Assign this integer values to the variable ye*
9. *Create an array of integer values varying from 1 to 31 and assign this array to the variable days*
10. *Initialise the variable ind_index to 1*
11. *Initialise the variable int_year to the content of the variable ys*
12. *Initialise the variable int_seas to1*
13. *Repeat the following instructions until the counter int_year is lower or equal to the content of the variable ye:*
    - *repeat the following instructions until the counter int_seas is lower or equal 3:*
      - *if int_seas is equal to 1 then:*
        - *create an array of integer values varying from 1 to 4*
        - *assign this array to the variable months*
      - *else if int_seas is equal to 2 then:*
        - *create an array of integer values varying from 5 to 8*
        - *assign this array to the variable months*
      - *else:*
        - *create an array of integer values varying from 9 to 12*
        - *assign this array to the variable months*
      - *find the indices of the rows of the mtx_icesat_f 2D array where: the elements mtx_icesat_f(*,4) are equal to the variable ind_year and the elements mtx_icesat_f(*,2) are members of the set given by the variable months and the elements mtx_icesat_f(*,3) are members of the set given by the variable days*
      - *save the result of this query into the 1D array ind_row*
      - *if the ind_row array is not empty then:*
        - *assign the content of mtx_icesat_f(ind_row,16) to the 1D array vec_X*
        - *assign the content of mtx_icesat_f(ind_row,15) to the 1D array vec_Y*
        - *assign the content of mtx_icesat_f(ind_row,18) to the 1D array variable vec_diff*
        - *compute the sum element by element of mtx_icesat_f(ind_row,10) and mtx_icesat_f(ind_row,12)*
        - *assign the result of the addition to the 1D array vec_ele*
        - *divide each element of mtx_icesat_f(ind_row,2) by 12 and add the result, element by element, to the elements of mtx_icesat(ind_row,4)*

- *divide each element of mtx_icesat_f(ind_row,3) by 365 and add the result, element by element, to the elements obtained in the previous addition*
- *compute the mean of the result obtained in the previous step*
- *save the result of the mean into the variable vec_time at the index specified by the variable ind_index*
- *initialise the counter i to 1*
- *initialise the counter j to 1*
- *repeat the following instructions until the counter i is lower or equal to the content of the variable nrow_d:*
  - *repeat the following instructions until the counter j is lower or equal to the content of the variable ncol_d:*
    - *assign the content of mtx_XX(i,j) to the variable centreX*
    - *assign the content of mtx_YY(i,j) to the variable centreY*
    - *find the indices where the following conditions are satisfied at the same time (logic and): the elements of the array vec_X are greater than the content of the variable centreX decreased by haft the content of the variable dx; the elements of vec_X are lower than the content of the variable centreX increased by half the content of the variable dx; the elements of vec_Y are greater than the content of the variable centreY decreased by haft the content of the variable dy; the elements of vec_Y are lower than the content of the variable centreY increased by half the content of the variable dy*
    - *save the result of this query into the 1D array vec_ind*
    - *if the array vec_ind is not empty then:*
      - *compute the mean of the elements of the array vec_diff at the indices specified in vec_ind*
      - *assign the result to the variable m_diff*
      - *compute the mean of the elements of the array vec_ele at the indices specified in vec_ind*
      - *assign the result to the variable m_ele*
      - *if m_ele is negative then assign zero to it*
      - *determine the number of elements of the array vec_ind*
      - *assign this number to the variable N*
    - *else:*
      - *assign NaN to the variable m_diff*
      - *assign NaN to the variable m_ele*
      - *assign 0 to the variable N*

- *save m_diff into the element of the 3D array mtx_DH identified by (ind_index,i,j)*
- *save m_ele into the element of the 3D array mtx_ele identified by (ind_index,i,j)*
- *save N into the element of the 3D array mtx_N identified by (ind_index,i,j)*
- *increment the counter j of 1*

– *increment the counter i of 1*

- *else:*
  - *assign NaN to mtx_DH(ind_index,*,*)*
  - *assign NaN to mtx_ele(ind_index,*,*)*
  - *assign 0 to mtx_N(ind_index,*,*)*
- *increment the counter ind_index of 1*
- *increment the counter ind_seas of 1*

- *increment the counter ind_year of 1*

5.2.6 Density and spatial filtering module

General description
This module applies density and spatial filters to each difference grid corresponding to each season, basically to remove unreliable measurements and outliers.

Definition and detailed description of the objects to be computed
The 3D arrays *mtx_DH* and *mtx_N*, whose first index corresponds to the time step and the second and the third define the spatial grid format, are checked in order to assign to the elements that do not satisfy the conditions given by the density and spatial filtering the special value of NaN (not a number). It is worth remembering that when the first index of the *mtx_DH* array is fixed, that is a specific season is considered, the corresponding 2D array represents the results of averaging the difference between the laser measured elevations and the DEM interpolated elevations over a grid cell space. Similarly, for the *mtx_N* array which the 2D array represents the number of the data points used within each grid cell for the averaging process at the time step specified by the fixed value of the first index.

The first simple filter to apply to both *mtx_DH* and *mtx_N* consists in checking for each season the number of measurements averaged within each grid cell. The check is to be considered failed when this number is lower than a fixed threshold provided by input (*N_th*).
The second filter to apply is again based on the inspection for each season of the number of measurements averaged within each grid cell. This time the check is to be considered failed when this number is greater than a fixed threshold provided by input (*N_xo_th*).

Then for each grid cell, the median of the temporal trend of the number of averaged measurements is computed and the seasons where this number deviates by more than a fixed threshold from the median are identified and discarded (i.e., for each grid cell the averaged difference and the corresponding number of measurements are set to NaN). The threshold of this filtering is provided in input (*med_th*).

Another statistical filtering is then applied. For each grid cell, the standard deviation from the mean value of the temporal trend of the averaged differences is computed. The seasons where

these differences deviate more than three times the standard deviation are discarded (i.e., for each grid cell the averaged difference and the corresponding number of measurements are set to NaN). Eventually, if the number of useful seasons is lower than a fixed number given as input (*min_points*), then all the seasons of the entire acquisition period are discarded (i.e., for each grid cell the averaged difference and the corresponding number of measurements in all the time step are set to NaN).

Definition of variables

Input variables:

- *mtx_DH* = 3D array where each time step is placed into different layer, corresponding to the first index of the array; each layer then represents the averaging results of the difference between the laser measured elevations and the DEM interpolated elevations; the average is applied for representing such quantity in a grid format (3D array of real elements, m);
- *mtx_N* = 3D array where each time step is placed into different layer, corresponding to the first index of the array; each layer then represents the number of the data points used within each grid cell for averaging both the laser measurements and the difference between the laser measured elevations and the DEM interpolated elevations (3D array of real elements);
- *nrow_d* = identifies the number of columns of the grid (scalar long integer, non-dimensional);
- *ncol_d* = identifies the number of rows of the grid (scalar long integer, non-dimensional);
- *min_points* = minimum number of valid epochs for computing the trend (scalar integer, non-dimensional);
- *N_th* = minimum number of measurements at one epoch in each grid cell (scalar integer, non-dimensional);
- *N_xo_th* = number of measurements withing a grid cell identifying a cross-over point (scalar integer, non-dimensional);
- *med_th* = median threshold (scalar integer, non-dimensional).

Internal variables:

- *vec_ind* = 1D array identifying indices where the condition for filtering are satisfied (1D array of integer elements, adimentional);
- *i* = counter (scalar long integer, non-dimensional);
- *j* = counter (scalar long integer, non-dimensional);
- *ind_not_NaN* = 1D array identifies indices where the *mtx_DH* presents useful data (1D array of integer elements, non-dimensional);
- *int_N_median* = median (scalar real, non-dimensional);
- *vec_DH* = 1D array identifying for each grid cell the averaging results of the difference between the laser measured elevations and the DEM interpolated elevations (1D array of real elements, m).

Output variables:

- *mtx_DH* = identifies the *mtx_DH* array after the filtering is applied, it has the same format of the *mtx_DH* provided as input (3D array of real elements. m);
- *mtx_N* = identifies the *mtx_N* array after the filtering is applied, it has the same format of the *mtx_N* provided as input (3D array of real elements).

Model equations and logical flow diagram

The mathematical aspect of this module is mainly based on the statistical definition of mean, median and standard deviation. The first two statistics are estimates of where the "middle" of a set of data is. The standard deviation is the average distance between the actual data and the mean. Their definition is reported below:

- Mean
  The mean (or average) which represents a calculated central value of a set on numbers, is obtained by dividing the sum of observed values by the number of observations, *n*. The formula is:

$$\overline{X} = \frac{\sum_{i=1}^{n} x_i}{n} \tag{5.17}$$

- Median
  The median of an observation variable is the value at the middle when the data is sorted in ascending order. It is an ordinal measure of the central location of the data values. If there is an odd number of data, the median value corresponds to the data that is in the middle, whereas if there is an even number of data, the middle pair must be averaged to find the median value. The median is often close to the mean value..

- Standard Deviation
  The standard deviation gives an idea of how close the entire set of data is to the average value. Data sets with a small standard deviation have tightly grouped data. Data sets with large standard deviations have data spread out over a wide range of values. The formula for standard deviation is given below:

$$\sigma = \left(\frac{1}{n-1} \sum_{i=1}^{n} \left(x_i - \overline{X} \right)^2 \right)^{\frac{1}{2}} \tag{5.18}$$

  The standard deviation (the square root of variance) of a sample can be used to estimate a population's true variance. Equation above is an unbias estimate of it.

In Fig. 5.8 the logical flow diagram is shown.

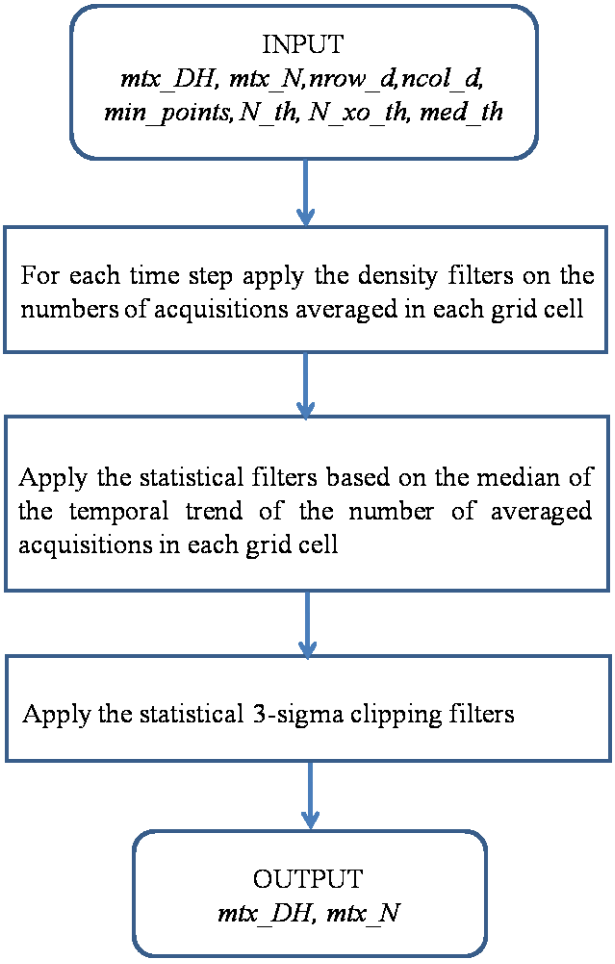

Fig. 5.8: Logical flow diagram of the Density and spatial filtering module.

Pseudo Code

1. *Find where the elements of the 2D array mtx_N are lower than the content of the variable N_th (provide the result as a linear indexing)*
1. *Save the result of the query in the 1D array vec_ind*
2. *Assign NaN to the elements of the 2D array mtx_DH identified by the linear indices saved in vec_ind*
3. *Assign NaN to the elements of the 2D array mtx_N identified by the linear indices saved in vec_ind*
4. *Find where the elements of the 2D array mtx_N are greater than the content of the variable N_xo_th (provide the result as a linear indexing)*
5. *Save the result of the query in the 1D array vec_ind*
6. *Assign NaN to the elements of the 2D array mtx_DH identified by the linear indices saved in vec_ind*
7. *Assign NaN to the elements of the 2D array mtx_N identified by the linear indices saved in vec_ind*
8. *Initialise the counter i to 1*
9. *Initialise the counter j to 1*
10. *Repeat the following instructions until the counter i is lower or equal to the content of the variable nrow_d:*

- *repeat the flowing instructions until the counter j is lower or equal to the content of the variable ncol_d:*
  - *find where the elements of mtx_DH(*,i,j) are not NaN (provide the result as a linear indexing)*
  - *save the result of this query in the 1D array ind_not_NaN*
  - *if the array ind_not_NaN is not empty then:*
    - *reshape mtx_N(ind_not_NaN,i,j) as a 1D row array with a number of elements equal to the length of the array ind_not_NaN*
    - *compute the median of the reshaped array*
    - *assign the result to the variable int_N_median*
    - *find where the elements of mtx_N(*,i,j) are lower than the difference between the content of the variable int_N_median and the content of the variable med_th*
    - *assign the result of this query to the 1D array vec_ind*
    - *assign NaN to mtx_DH(vec_ind,i,j)*
    - *assign NaN to mtx_N(vec_ind,i,j)*
    - *find where the elements of mtx_N(*,i,j) are greater than the sum of the content of the variable int_N_median and the content of the variable med_th*
    - *assign the result of this query to the 1D array vec_ind*
    - *assign NaN to mtx_DH(vec_ind,i,j)*
    - *assign NaN to mtx_N(vec_ind,i,j)*
    - *assign the elements mtx_DH(*,i,j) to the 1D array vec_DH*
    - *subtract to each element of the array vec_DH its mean value*
    - *find where the elements of the vec_DH array have the absolute value greater than three times the standard deviation computed using the elements of the array*
    - *assign the result of this query to the 1D array vec_ind*
    - *if the array vec_ind is not empty then:*
      - *assign NaN to the elements vec_DH(vec_ind)*
      - *assign NaN to the elements mtx_DH(vec_ind,i,j)*
      - *assign NaN to the elements mtx_N(vec_ind,i,j)*
    - *if the sum of the finite elements of the array vec_DH is lower than the content of the variable min_points then:*
      - *assign NaN to the elements mtx_DH(*,i,j)*
      - *assign NaN to the elements mtx_N(*,i,j)*
  - *increment the counter j of 1*
- *increment the counter i of 1*

5.2.7 Altimetry elevation trend and error analysis module

General description

This module derives elevation trends from the difference between the laser measured elevations and the DEM interpolated elevations over the laser corresponding acquisition period and assesses the accuracy of the estimation.

Definition and detailed description of the objects to be computed

For each grid cell, identified by the indices (*i,j*), the values of the 3D arrays *mtx_DH(*,i,j)*, representing the results of averaging the difference between the laser measured elevations and the DEM interpolated elevations over each time step, are linearly interpolated in order to estimate the elevation trend, using the least square method. Once the coefficients of the linear equations are determined, the accuracy of the estimation is assessed by computing the confidence interval corresponding to a given significance level (*alpha*), which represents the probability of the error committed in rejecting the assumption that the true elevation trend falls within the interval around the estimated elevation trend.

Definition of variables

Input variables:

- *mtx_DH* = 3D array where each time step is placed into different layer, corresponding to the first index of the array; each layer then represents the results of averaging the difference between the laser measured elevations and the DEM interpolated elevations and of the filtering described in the Density and spatial filtering module (3D array of real elements, m);
- *vec_time* = 1D array representing the mean time at each time step (1D array of real elements, year);
- *nrow_d* = identifies the number of columns of the grid (scalar long integer, non-dimensional)*;*
- *ncol_d* = identifies the number of rows of the grid (scalar long integer, non-dimensional);
- *N_er_th* = error threshold (scalar integer, m/yr);
- *mtx_ele* = 3D array where each time step is placed into different layer, corresponding to the first index of the array; each layer then represents the results of averaging the laser measured elevations; the average is applied for representing the elevation in a grid format (3D array of real elements, m);
- *alpha* = significance level representing the probability of the error committed in rejecting the assumption that the true elevation trend falls within an interval (the confidence interval) around the estimated elevation trend (scalar real, non-dimensional).

Internal variables:

- *i* = counter (scalar long integer, non-dimensional);
- *j* = counter (scalar long integer, non-dimensional);
- *k* = counter (scalar long integer, non-dimensional);
- *ind_not_NaN* = 1D array identifies indices where the *mtx_DH(*,i,j)* presents useful data (i.e., finite values), or in other words the useful time steps (or seasons) to consider for the trend evaluation (1D array of integer elements, non-dimensional);
- *ind_er* = 1D array identifies indices where the elements of the 2D array *mtx_error* assume value greater than the error threshold (1D array of integer elements, non-dimensional);
- *n* = number of elements of the 1D array *ind_not_NaN* (scalar integer, non-dimensional);

- *vec_DH* = 1D array of 1x*n* elements identifying for each grid cell the averaging results of the difference between the laser measured elevations and the DEM interpolated elevations at the useful seasons for the trend evaluation (1D array of real elements, m);
- *vec_tmp_time* = 1D array of 1x*n* elements containing the mean time of the time steps (or seasons) effectively used for the trend evaluation, that is the time steps where only finite value of the data are available (1D array of real elements, yr);
- *y* = 1D array of *n*x1 elements representing the array transpose of *vec_DH* (1D array of real elements, m);
- *X* = 2D array of *n*x2 elements: each element of the first column is 1, whereas the element of the *i-th* row of the second column corresponds to the *i-th* element of the 1D array *vec_tmp_time* (2D array of *n*x2 real elements);
- *Q* = 2D array of *n*x2 elements identifying the unitary matrix of the orthogonal-triangular decomposition of the 2D array *X* using the QR factorization algorithm in its "economy-size" version (2D array of *n*x2 real elements);
- *R* = 2D array of 2x2 elements identifying the upper triangular matrix of the orthogonal-triangular decomposition of the 2D array *X* using the QR factorization algorithm in its "economy-size" version (2D array of *2*x2 real elements);
- *E* = 1D array of 2 elements identifying the permutation vector of the the orthogonal-triangular decomposition of the 2D array *X* using the QR factorization algorithm in its "economy-size" version (1D array of integer elements);
- *nindcol* = number of independent columns of the 2D array *X* (scalar integer, non-dimensional);
- *s* = accumulator (scalar real, non-dimensional);
- *b* = 1D array of 2x1 elements representing the estimate of the coefficients of the interpolation line (1D array of real elements);
- *RI* = 2D array of 2x2 elements identifying the inverse of the 2D array *R* (2D array of *2*x2 real elements)
- *ndegree* = number of degrees of freedom (scalar integer, non-dimensional);
- *yhat* = 1D array of *n*x1 elements representing the predicted elevation differences at each mean time (1D array of real elements);
- *r* = 1D array of *n*x1 elements representing the residuals between the elevation differences, obtained between the laser measurements and the DEM interpolated elevations, and the predicted elevation differences (1D array of real elements);
- *normr* = scalar variable representing the norm of the 1D array of the residuals, *r* (scalar real);
- *rmse* = scalar variable representing the root mean square error of the interpolation (scalar real);
- *x* = scalar variable representing the inverse of the Student's *t* cumulative distribution function (scalar real);
- *xn* = scalar variable representing the inverse of the normal cumulative distribution function (scalar real);

- *p* = scalar variable representing the probability representing the chance that the interpolated curve is true; it is given by 1-*alpha*/2 (scalar real);
- *q* = scalar variable representing the probability given by *p*-0.5 (scalar real);
- *t* = logical variable (scalar integer, non-dimensional);
- *oneminusz* = scalar variable representing the inverse of the incomplete beta function (scalar real);
- *z* = scalar variable representing 1-*oneminusz* (scalar real);
- *mu* = scalar variable representing the mean of a normal distribution function (scalar real);
- *sigma* = scalar variable representing the standard deviation of a normal distribution function (scalar real);
- *x0* = scalar variable identifying the inverse of the complementary error function (scalar real);
- *se* = 1D array of 2x1 elements identifying the standard error (1D array of real elements);
- *sr* = 1D array of 2x1 elements; the *i-th* element represents the sum of the elements contained in the *i-th* row of the 2D array *RI* (1D array of real elements);
- *bint* = 2D array of 2x2 elements; the first column contains lower confidence bounds for each of the 2 coefficient estimated (the elements of the 1D array *b*, whereas the second column contains the upper confidence bounds (2D array of real elements).

Output variables:

- *mtx_ele_aver* = 2D array where each element *mtx_ele_aver*(*i,j*) corresponds to the average over the time seasons (temporal average) of the elevations resulting from the average of the laser measurements whose locations are inside a given grid cell (spatial average); in other words each element *mtx_ele_aver*(*i,j*) corresponds to the average of the elements *mtx_ele*(*,*i,j*) (2D array of real elements, m);
- *mtx_trend* = 2D array representing the elevation trend for each grid cell (2D array of real elements, m/year);
- *mtx_error* = 2D array representing the trend error at each grid cell (2D array of real elements, m/year).

Model equations and logical flow diagram

The mathematics of the module is mainly based on the theory of linear interpolation and regression analysis. As explained in the model equation and logical flow diagram of sections 5.3.4 and 5.3.5, for a given ICESat operation period the differences between the GLAS measured elevations and the DEM interpolated elevations (equation (5.13)) are averaged in order to represent the variation of the elevation on grid cells of a specific size (the latter is determined by multiplying the DEM resolution for the downscale factor, down_factor):

$$\overline{\Delta h_{\tilde{i}}}\left(x_c, y_c\right) = \frac{\sum_{k=1}^{N} \Delta h_{t_k^m - t_{REF}}\left(x_k^m, y_k^m\right)}{N} = \frac{\sum_{k=1}^{N} h_k^m - h_k' + \Delta h_k}{N} \tag{5.19}$$

where N is the number of laser acquisitions within a fixed range distance from the centre of each grid cell, (x_c, y_c), and t_k^m is the time of the each acquisition within the epoch considered, that is $t_k^m \in [t_{ini}, t_{fin}]$. The sub-index $\bar{t}$ is the mean time value associated with the averaged difference $\overline{\Delta h}$:

$$\bar{t} = \frac{\sum_{k=1}^{N} t_k^m}{N} \tag{5.20}$$

This procedure is repeated for each ICESat operational period and it provides the *mtx_DH* 3D array.
The elevation trend for each grid cell is determined by fitting a first order polynomial:

$$\Delta h(t) = b_0 + b_1 t \tag{5.21}$$

The slope of the polynomial, the coefficient b_1, represents the elevation trend in the data bin considered (Rinne et al., 2011).
Analytically, if $\bar{t}_i$ represents the averaged time of the *i-th* operational period, or epoch, and $\overline{\Delta h}_i(x_c, y_c)$ is the average difference relative to such time in the grid cell centred around (x_c, y_c), the first order polynomial is identified applying the least square method. The best estimation of the coefficients b_0 and b_1 is obtained minimising the sum square error:

$$\min_{b_0, b_1} \sum_{i=1}^{M} \left[\Delta h(\bar{t}_i) - \overline{\Delta h}_i\right]^2 = \min_{b_0, b_1} \sum_{i=1}^{M} \left[b_0 + b_1 \bar{t}_i - \overline{\Delta h}_i\right]^2 \tag{5.22}$$

where M is the total number of operational period, or epoch.
If the cost function $f(b_0, b_1)$ is defined as follow:

$$f(b_0, b_1) = \sum_{i=1}^{M} \left[b_0 + b_1 \bar{t}_i - \overline{\Delta h}_i\right]^2 \tag{5.23}$$

minimising the eq. (5.21) corresponds to solve the following system of equations:

$$\begin{cases} \dfrac{\partial f}{\partial b_0} = 0 \\ \dfrac{\partial f}{\partial b_1} = 0 \end{cases} \tag{5.24}$$

whose solutions are:

$$\hat{b}_0 = \frac{\sum_{i=1}^{M} \overline{\Delta h_i}}{M} - \hat{b}_1 \frac{\sum_{i=1}^{M} \bar{t}_i}{M} \tag{5.25}$$

$$\hat{b}_1 = \frac{\frac{\sum_{i=1}^{M} \bar{t}_i \overline{\Delta h_i}}{M} - \frac{\sum_{i=1}^{M} \bar{t}_i}{M} \frac{\sum_{i=1}^{M} \overline{\Delta h_i}}{M}}{\frac{\sum_{i=1}^{M} \bar{t}_i^{\,2}}{M} - \left(\frac{\sum_{i=1}^{M} \bar{t}_i}{M} \right)^2} \tag{5.26}$$

In order to understand if $\hat{b}_0$ and $\hat{b}_1$ provide a good estimation of the coefficients b_0 and b_1 of the linear equation (5.21), it is necessary to consider the average difference, $\overline{\Delta h_i}$, and the two estimators $\hat{b}_0$ and $\hat{b}_1$ as random variables. In particular, it is necessary to analyse how the latter distribute around their respectively exact values, b_0 and b_1.
It is demonstrable that (Draper and Smith, 1981):

1) $\mathrm{E}\{\hat{b}_0\} = b_0$, that is $\hat{b}_0$ is an unbiased estimator of b_0;
2) $\mathrm{E}\{\hat{b}_1\} = b_1$, that is $\hat{b}_1$ is an unbiased estimator of b_1;
3) $\mathrm{var}\{\hat{b}_0\} = \frac{\sum_{i=1}^{M} \bar{t}_i^{\,2}}{M \sum_{i=1}^{M} \left(\bar{t}_i - E\{\bar{t}_i\}\right)^2} \sigma^2$, where σ^2 is the variance of $\overline{\Delta h_i}$;
4) $\mathrm{var}\{\hat{b}_1\} = \frac{\sigma^2}{\sum_{i=1}^{M} \left(\bar{t}_i - E\{\bar{t}_i\}\right)^2}$, where σ^2 is the variance of $\overline{\Delta h_i}$.

The method of the confidence interval is introduced in this module for the determination of the error associated with the estimates.
As regards the angular coefficient of the linear interpolation, the method consists of determining the interval around $\hat{b}_1$ that is likely to contain the true parameter value, i.e. b_1, with a certain probability (called confidence level). In formula, the following expression defines the confidence bounds:

$$P\left(\left|\hat{b}_1 - b_1\right| \le \delta\right) = P\left(\hat{b}_1 - \delta \le b_1 \le \hat{b}_1 + \delta\right) = 1 - \alpha \tag{5.27}$$

where δ is half of the interval extent, 1 - α is the confidence level, that is the probability of producing an interval containing b_1, and α is the significance level (it corresponds to the input variable *alpha*), that is the probability that the true parameter value, b_1, is outside the confidence interval (or, in other words, it is the probability of the error committed in rejecting the assumption that the true parameter value, b_1, falls within the confidence interval).

In the assumption that the average differences $\overline{\Delta h}_i$ are normally distributed, it follows that also $\hat{b}_1$ distributes normally, because it is a linear combination of the random variables $\overline{\Delta h}_i$ (see equation, 5.26). Hence, the random variable:

$$Z = \frac{\hat{b}_1 - b_1}{\sqrt{\frac{\sigma^2}{\sum_{i=1}^{M}(\bar{t}_i - E\{\bar{t}_i\})^2}}} = \frac{\hat{b}_1 - b_1}{\left[\mathrm{var}(\hat{b}_1)\right]^{\frac{1}{2}}} \tag{5.28}$$

has a normal distribution with mean equal to 0 and variance equal to 1. If the variance σ^2 is known, then it is possible to use the Sheppard tables (Sheppard, 1903; Sheppard, 1939) to determine the confidence interval, that is to determine the value of $z_{1-\alpha/2}$ that satisfies the following expression:

$$\begin{aligned} P\left(-z_{1-\alpha/2} \le Z \le z_{1-\alpha/2}\right) &= P\left(-z_{1-\alpha/2} \le \frac{\hat{b}_1 - b_1}{\left[\mathrm{var}(\hat{b}_1)\right]^{\frac{1}{2}}} \le z_{1-\alpha/2}\right) = \\ &= P\left(\hat{b}_1 - z_{1-\alpha/2}\left[\mathrm{var}(\hat{b}_1)\right]^{\frac{1}{2}} \le b_1 \le \hat{b}_1 + z_{1-\alpha/2}\left[\mathrm{var}(\hat{b}_1)\right]^{\frac{1}{2}}\right) = 1-\alpha \end{aligned} \tag{5.29}$$

From the above equation, the half extent of the confidence interval is given by:

$$\delta = z_{1-\alpha/2}\left[\mathrm{var}(\hat{b}_1)\right]^{\frac{1}{2}} = z_{1-\alpha/2}\sqrt{\frac{\sigma^2}{\sum_{i=1}^{M}(\bar{t}_i - E\{\bar{t}_i\})^2}} \tag{5.30}$$

In most of the case, the variance σ^2 is unknown and it is often estimated by means of the following expression adapted to the symbology defined and used above:

$$s^2 = \frac{1}{M-2}\sum_{i=1}^{M}\left[\overline{\Delta h}_i - \left(\hat{b}_0 + \hat{b}_1\bar{t}_i\right)\right]^2 \tag{5.31}$$

When s^2 is used to replace the variance σ^2, $\hat{b}_1$ does not present anymore a normal distribution. However, the variable:

$$t = \frac{\hat{b}_1 - b_1}{\sqrt{\frac{s^2}{\sum_{i=1}^{M}(\bar{t}_i - E\{\bar{t}_i\})^2}}} \tag{5.32}$$

assumes a *t*-Student distribution with $(M-2)$ degrees of freedom (the 2 degrees of freedom subtracted to the number M of couples $(\bar{t}_i, \overline{\Delta h}_i)$ is due to the number of degrees of freedom on which the estimate s^2 is based).
Indicating with $t_{1-\alpha/2}$ the upper critical value of the *t*-Student distribution with *M*-2 degrees of freedom which identifies the tail of the distribution with a subtended area equal to the probability of $\alpha/2$, the following expression allows the determination of the half extent of the confidence interval:

$$P\left(-t_{1-\alpha/2} \le t \le t_{1-\alpha/2}\right) = P\left(-t_{1-\alpha/2} \le \frac{\hat{b}_1 - b_1}{\sqrt{\frac{s^2}{\sum_{i=1}^{M}\left(\bar{t}_i - E\{\bar{t}_i\}\right)^2}}} \le t_{1-\alpha/2}\right) = \tag{5.33}$$

$$= P\left(\hat{b}_1 - t_{1-\alpha/2}\sqrt{\frac{s^2}{\sum_{i=1}^{M}\left(\bar{t}_i - E\{\bar{t}_i\}\right)^2}} \le b_1 \le \hat{b}_1 + t_{1-\alpha/2}\sqrt{\frac{s^2}{\sum_{i=1}^{M}\left(\bar{t}_i - E\{\bar{t}_i\}\right)^2}}\right) = 1-\alpha$$

which is given by:

$$\delta = t_{1-\alpha/2}\sqrt{\frac{s^2}{\sum_{i=1}^{M}\left(\bar{t}_i - E\{\bar{t}_i\}\right)^2}} = t_{1-\alpha/2}\, s_{\hat{b}_1} \tag{5.34}$$

where

$$s_{\hat{b}_1} = \sqrt{\frac{s^2}{\sum_{i=1}^{M}\left(\bar{t}_i - E\{\bar{t}_i\}\right)^2}} \tag{5.35}$$

denotes the standard error of the estimate.
Summarising, the confidence interval of the true elevation trend, b_1, corresponding to a confidence level of 1-α, can be written as follow:

$$\left[\hat{b}_1 - t_{1-\alpha/2}s_{\hat{b}_1}, \hat{b}_1 + t_{1-\alpha/2}s_{\hat{b}_1}\right] \tag{5.36}$$

In similar way, it is possible to compute the confidence interval of the intercept of the linear interpolation, i.e. b_0. The half extend is given by the following expression:

$$\delta = t_{1-\alpha/2}\sqrt{\frac{\sum_{i=1}^{M}\bar{t}_i^{\,2}}{M\sum_{i=1}^{M}\left(\bar{t}_i - E\{\bar{t}_i\}\right)^2}}\,s = t_{1-\alpha/2}\,s_{\hat{b}_0} \tag{5.37}$$

And consequently the full confidence interval can be written as:

$$\left[\hat{b}_0 - t_{1-\alpha/2}s_{\hat{b}_0}, \hat{b}_0 + t_{1-\alpha/2}s_{\hat{b}_0}\right] \tag{5.38}$$

In both the equations (5.36) and (5.37), $t_{1-\alpha/2}$ is the $100\left(1-\frac{\alpha}{2}\right)$ percentage point of a *t*-Student distribution with $(M-2)$ degree of freedom. It can be determined by means of *t*-Student tables. However, when the number of degrees of freedom assumes a value lower than 1000, the $t_{1-\alpha/2}$ is obtained by using the inverse incomplete beta function (Abromowitz & Stegun, 1972), which computes the inverse of that function with respect to the integration limit using Newton's method. When it assumes a bigger value, the asymptotic expansion of the inverse function of a *t*-student distribution is applied (formula number 26.7.5 reported in the Abromowitz & Stegun, 1972).
When the confidence interval of the elevation trend is determined, the trend error is set to be the greatest value between the lower and upper bound of the confidence interval, that is:

$$error = \max\left(\left|\hat{b}_1 - \left(\hat{b}_1 - t_{1-\alpha/2}s_{\hat{b}_1}\right)\right|, \left|\hat{b}_1 - \left(\hat{b}_1 + t_{1-\alpha/2}s_{\hat{b}_1}\right)\right|\right) \tag{5.39}$$

In Fig. 5.9 the logical flow diagram is shown.

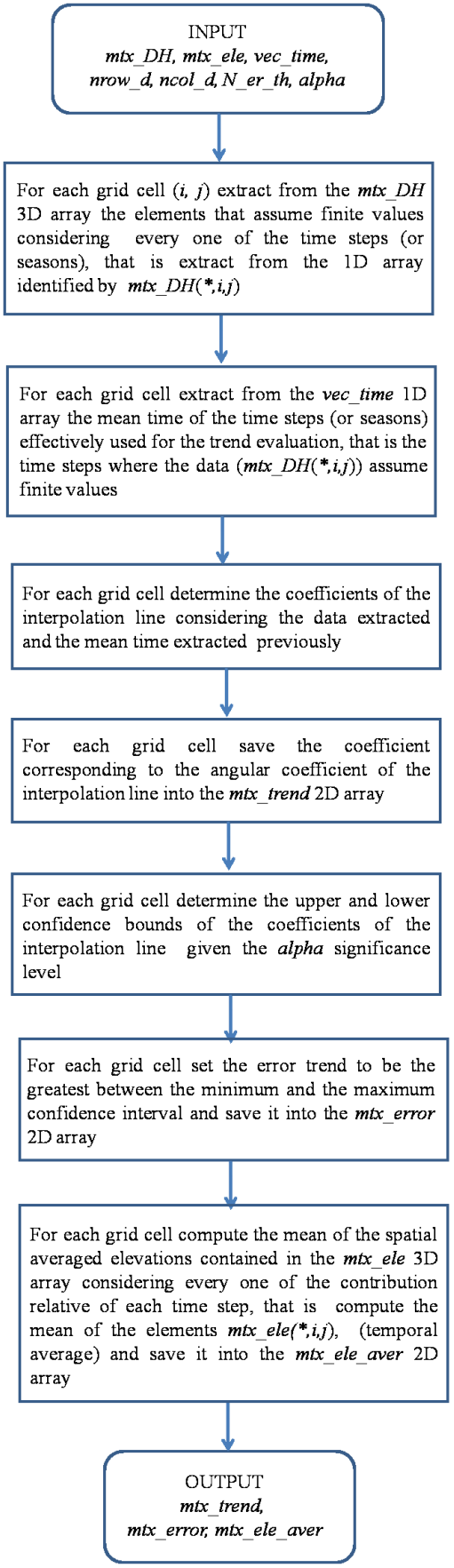

Fig. 5.9: Logical flow diagram of the Altimetry elevation trend and error analysis module.

Pseudo Code

The functions marked in green are support modules that are described in section 5.2.9.

1. *Initialise the counter i to 1*
2. *Initialise the counter j to 1*
3. *Repeat the following instructions until the counter i is lower or equal to the content of the variable nrow_d:*
    - *repeat the following instructions until the counter j is lower or equal to the content of the variable ncol_d:*
        - *find where the elements of mtx_DH(*,i,j) are not NaN (provide the result as a linear indexing)*
        - *save the result of this query in the 1D array ind_not_NaN*
        - *if the array ind_not_NaN is not empty then:*
            - *assign the elements mtx_DH(ind_not_NaN,i,j) to the 1D array vec_DH*
            - *find the number of the elements of the 1D array ind_not_NaN*
            - *assign the result to the variable n*
            - *extract from the 1D array vec_time the elements corresponding to the indices saved in the 1D array ind_not_NaN*
            - *save these elements in the 1D array vec_tmp_time*
            - *subtract to each element of the array vec_DH its mean value*
            - *compute the transpose of the resulting vec_DH array*
            - *assign the result to the 1D array y*
            - *create the 2D array X having a number of rows corresponding to the value of the variable n and a number of columns equal to 2; each element of the first column is 1, whereas the element of the i-th row of the second column corresponds to the i-th element of the 1D array vec_tmp_time*
            - *call the support module named QR decomposition passing the variable X*
            - *assign the first output of the function to the 2D array Q*
            - *assign the second output of the function to the 2D array R*
            - *assign the third output of the function to the 1D array E*
            - *initialise the variable s to 0*
            - *initialise the counter k to 1*
            - *repeat the following instructions until the counter k is lower or equal to 2*
                - *if the absolute value of the element R(k,k) is greater than 10e-13 multiplied by the maximum between n and 2 then*
                    - *increment the variable s of 1*
            - *assign the content of the variable s to the variable nindcol*
            - *if nindcol is less than 2 then*
                - *keep the first nindcol rows and the first nindcol columns of the 2D array R (i.e., R = R (1:nindcol,1:nindcol))*

- *keep the nindcol first columns of the 2D array Q (i.e., Q = Q (*,1:nindcol))*
- *keep the first nindcol elements of the 1D array E (i.e., E = E(1:nindcol))*

- *initialise all the elements of the 1D array b to 0*
- *compute the backslash or matrix left division between the 2D array R and the 1D array resulting from the matrix multiplication between the transpose of the 2D array Q and the 1D array y (i.e., R \ (Q'*y)) (this corresponds to compute the inverse of the 2D array R and then to compute the matrix multiplication between the result of such inversion and the result of the matrix multiplication between the transpose of the 2D array Q and the 1D array y)*
- *assign the result of the backslash division to the 1D array b*
- *permute the rows of the 1D array b based on the order given by the elements of the 1D array E (i.e., b=b(E))*
- *compute the inverse of the 2D array R*
- *assign the result to the 2D array RI*
- *compute the maximum between zero and the content of the variable n decreased by the content of the variable nindcol*
- *assign the result to the variable ndegree*
- *compute the matrix multiplication between the 2D array X and the 1D array b*
- *assign the result of this computation to the 1D array yhat*
- *compute the difference, element to element, between the 1D array y and the 1D array yhat*
- *assign the result of this computation to the 1D array r*
- *call the support module named norm passing the variable r*
- *assign the result of the function to the variable normr*
- *if ndegree is different from 0 then*
  - *divide the variable normr by the square root of the variable ndegree*
  - *assign the result to the variable rmse*
  - *divide the content of the variable alpha by 2*
  - *subtract the result to 1*
  - *assign the result of the subtraction to the variable p*
  - *assign NaN to the variable x*
  - *if p is equal to 0 and ndegree is greater than 0 then assign –Inf to the variable x*
  - *if p is equal to 1 and ndegree is greater than 0 then assign Inf to the variable x*
  - *if p assumes a value between 0 and 1 and ndegree is equal to 1 then*
    - *subtract 0.5 to the variable p*
    - *multiply the result for pi greek*
    - *compute the tangent of the result of the multiplication*
    - *assign the result to the variable x*

- *if p assumes a value between 0 and 1 and ndegree is less than 1000 then*
  - *subtract 0.5 to the variable p*
  - *assign the result to the variable q*
  - *if the absolute value of the variable q is less than 0.25 then assign 1 to the variable t, otherwise assign 0 to the same variable*
  - *assign 0 to the variable z*
  - *assign 0 to the variable oneminusz*
  - *if t is equal to 1 then*
    - *call the betaincinv support module passing as parameters the following quantities in the order here specified: two times the absolute value of the variable q; 0.5 and the variable ndegree divided by 2*
    - *assign the output of the betaincinv module to the variable oneminusz*
    - *subtract the variable oneminusz to 1*
    - *assign the result to the variable z*
  - *else*
    - *call the betaincinv support module passing as parameters the following quantities in the order here specified: 1 minus two times the absolute value of the variable q; the variable ndegree divided by 2 and 0.5*
    - *assign the output of the betaincinv module to the variable z*
    - *subtract the variable z to 1*
    - *assign the result to the variable oneminusz*
  - *compute the ratio between the variables oneminusz and z*
  - *multiply the result for the variable ndegree*
  - *compute the square root of the above multiplication*
  - *multiply the result for the sign of the variable q*
  - *assign the result to the variable x*
- *if p assumes a value between 0 and 1 and ndegree is greater than 1000 then*
  - *assign 0 to the variable mu*
  - *assign 1 to the variable sigma*
  - *call the support module named inverse erfc passing the variable p multiplied by 2*
  - *assign the output of the function to the variable x0*
  - *multiply the variable x0 for the square root of 2*
  - *multiply the result for -1*

- *multiply the results for the variable sigma*
- *add to the result of the above multiplication the value of the variable mu*
- *assign the result to the variable xn*
- *implement the Abromowitz & Stegun formula knowing the value of the variaqble xn and ndegree*
- *assign the result to the variable x*

- *else*
  - *assign NaN to the variable rmse*
  - *assign 0 to the variable x*
- *initialise all the elements of the 1D array se to 0*
- *compute the square of each element of the 2D array RI*
- *for each row of the resulting 2D array compute the sum of the elements contained in such row*
- *assign the result to the 1D array sr*
- *compute the square root of each element of the 1D array sr*
- *multiply each element of the resulting 1D array by the variable rmse*
- *assign the result of the multiplication to the 1D array se*
- *permute the rows of the 1D array se based on the order given by the elements of the 1D array E (i.e., se=se(E))*
- *multiply each element of the 1D array se by x*
- *subtract element by element the resulting 1D array to the 1D array b*
- *assign the resulting 1D array to the first column of the 2D array bint*
- *multiply each element of the 1D array se by x*
- *add element by element the resulting 1D array to the 1D array b*
- *assign the resulting 1D array to the second column of the 2D array bint*
- *assign the element b(2) to mtx_trend(i,j)*
- *compute the absolute value of the difference between the element b(2) and the element bint(2,1)*
- *compute the absolute value of the difference between the element b(2) and the element bint(2,2)*
- *find the maximum between these two absolute value differences*
- *assign the result to mtx_error(i,j)*

- *else*
  - *assign NaN to mtx_trend(i,j)*
  - *assign NaN to mtx_error(i,j)*
  - *compute the mean of mtx_ele(*,i,j)*
- *assign the result to mtx_ele_aver*

4. *Find where the elements of the 2D array mtx_error are greater than the content of the variable N_er_th (provide the result as a linear indexing)*
5. *Save the result of the query in the 1D array ind_er*
6. *Assign NaN to the elements of the 2D array mtx_trend identified by the linear indices saved in ind_er*

7. *Assign NaN to the elements of the 2D array mtx_error identified by the linear indices saved in ind_er*

5.2.8 DEM mask module

General description
This module masks out the results of the averaged elevations, the estimated elevation trends and the relative errors obtained over grid cells located outside the glacier mask.

Definition and detailed description of the objects to be computed
The DEM mask used is based on the DEM provided as input. In particular, the mask outlines are identified by those grid cell where the DEM elevation is equal to 0 (ocean mask).

Definition of variables

Input variables:

- *mtx_XX* = identifies the result of the down-sampling applied to the 2D array corresponding to *mtx_DEM(1,*,*)* (2D array of real elements, m);
- *mtx_YY* = identifies the result of the down-sampling applied to the 2D array corresponding to *mtx_DEM(2,*,*)* (2D array of real elements, m);
- *mtx_DEM_h* = identifies the result of the down-sampling applied to the 2D array corresponding to *mtx_DEM(3,*,*)* (2D array of real elements, m);
- *mtx_ele_aver* = 2D array where each element *mtx_ele_aver*(*i,j*) corresponds to the average over the time seasons (temporal average) of the elevations resulting from the average of the laser measurements whose locations are inside a given grid cell (spatial average); in other words each element *mtx_ele_aver*(*i,j*) corresponds to the average of the elements *mtx_ele*(*,*i,j*) (2D array of real elements, m);
- *mtx_trend* = 2D array representing the elevation trend for each grid cell (2D array of real elements, m/year);
- *mtx_error* = 2D array representing the trend error at each grid cell (2D array of real elements, m/year).

Internal variables:

- *ind* = 1D array identifies indices where the *mtx_DEM* assumes values equal to 0 (1D array of integer elements, non-dimensional).

Output variables:

- *mtx_XX_mask* =2D array obtained from the 2D array *mtx_XX* where the elements identified by grid cells with a DEM elevation of 0 m have been set to NaN (2D array of real elements, m)
- *mtx_YY_mask* =2D array obtained from the 2D array *mtx_YY* where the elements identified by grid cells with a DEM elevation of 0 m have been set to NaN (2D array of real elements, m);

- *mtx_ele_aver_mask* =2D array obtained from the 2D array *mtx_ele_aver* where the elements identified by grid cells with a DEM elevation of 0 m have been set to NaN (2D array of real elements, m);
- *mtx_trend_mask* = 2D array obtained from the 2D array *mtx_trend* where the elements identified by grid cells with a DEM elevation of 0 m have been set to NaN (2D array of real elements, m/year);
- *mtx_error_mask* = 2D array obtained from the 2D array *mtx_error* where the elements identified by grid cells with a DEM elevation of 0 m have been set to NaN (2D array of real elements, m/year).

Model equations and logical flow diagram

This module performs a query over the DEM elevation dataset (*mtx_DEM*(3,*,*)) for finding the indices (*i,j*) corresponding to an elevation of 0 m. Consequently, no analytical equations are implemented.

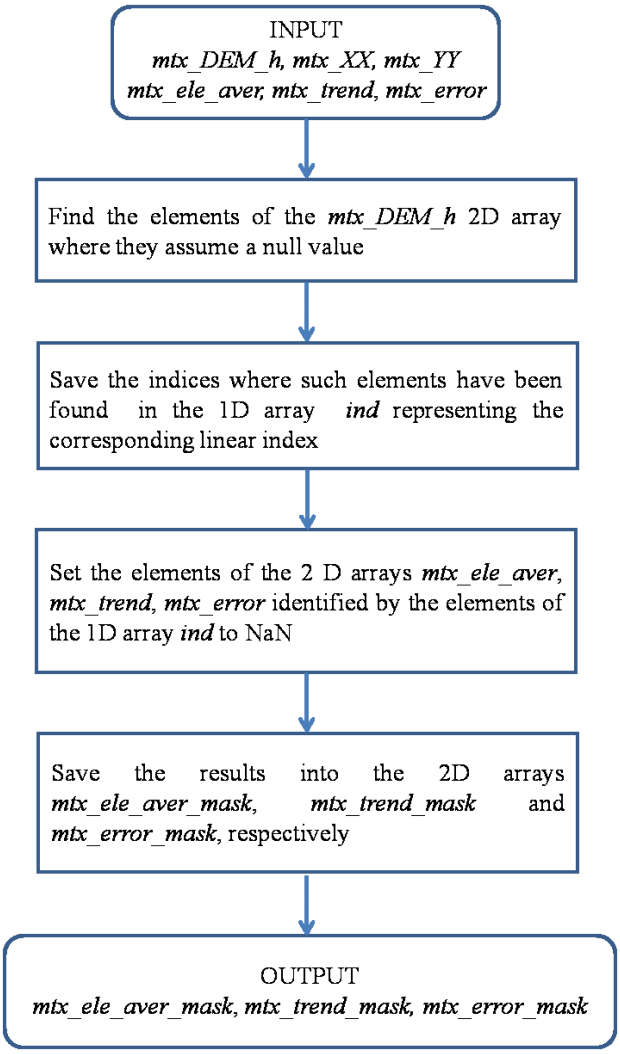

Fig. 5.10: Logical flow diagram of the Glacier mask module.

Pseudo Code

1. *Find where the elements of the 2D array mtx_DEM_h are equal to 0*
2. *Save the results of the query in the 1D array ind*
3. *Assign NaN to the elements of the 2D array mtx_XX identified by the linear indices saved in ind and save the result into the 2D array mtx_XX_mask*
4. *Assign NaN to the elements of the 2D array mtx_YY identified by the linear indices saved in ind and save the result into the 2D array mtx_YY_mask*
5. *Assign NaN to the elements of the 2D array mtx_ele_aver identified by the linear indices saved in ind and save the result into the 2D array mtx_ele_aver_mask*
6. *Assign NaN to the elements of the 2D array mtx_trend identified by the linear indices saved in ind and save the result into the 2D array mtx_trend_mask*
7. *Assign NaN to the elements of the 2D array mtx_error identified by the linear indices saved in ind and save the result into the 2D array mtx_error_mask*

5.2.9 Support modules

Four elementary modules are called in support of the repeat-altimetry macro-modules; *qr* performs orthogonal triangular decomposition, *norm* computes vector and matrix norms, *betaincinv* computes the beta inverse incomplete distribution function, and *erfcinv* computes the inverse complementary error function. These support modules provide low-level functionality, and are common features of numerical linear least-squares solutions for curve fitting. The support modules have been designed to implement the same functionality as the *qr*, *norm*, *betacinv* and *erfcinv* Matlab routines on which they are based.

Orthogonal triangular decomposition

The *qr* support module performs orthogonal triangular decomposition, and is indirectly employed in the linear regression calculation of elevation trends. Given an M x N matrix, the procedure computes the QR decomposition (factorisation), which is useful in solving least squares applications. The QR factorisation produces two matrices, Q and R, such that the solution is the matrix multiplication of Q and R. The technique is to first compute the factorisation, which yields the orthogonal matrix Q and the upper triangular matrix R, then the solution vector is computed. The input variable is the matrix to be decomposed, and the output variables are the triangular matrix (R) and the unitary matrix (Q) whose product equals the input matrix.

Vector and matric norms

The *norm* support module computes the norm of the input variable, and is indirectly employed in the linear regression calculation of elevation trends. In linear algebra, functional analysis and related areas of mathematics, a norm is a function that assigns a strictly positive length or size to each vector. In the case of vectors, this is equal to the Euclidean distance, in the case of matrices, this is equal to the largest singular value. The input variable is the matrix, and the output variable is the scalar norm of the input variable.

Inverse incomplete beta distribution function

The *betaincinv* support module computes the inverse incomplete beta function, and is indirectly employed in the linear regression calculation of elevation trends. Specifically, the function is called as part of the determination of confidence intervals on the output regression parameters. The input variables to the module are the parameters specifying the beta function

and the output variable is the value of the inverse incomplete beta function. *betaincinv* computes the inverse of the incomplete beta function with respect to the integration limit using Newton's method.

Inverse complementary error function

The *erfcinv* support module computes the inverse complementary error function, also known as the Gauss inverse complementary error function, and is indirectly employed in the linear regression calculation of elevation trends. The error function is a special function of sigmoid shape occuring in probability theory and statistics, and is used to compute accurately and efficiently the inverse cumulative probabilities of normal distribution. The *erfcinv* function is specifically called to aid the computation of confidence intervals. The input variable is an expression for which the inverse complementary error function is to be evaluated, and the output variable is the inverse complementary error function.

6. Velocity from optical sensors

6.1 General macro-module description

The macro-module for optical images begins with an optional module for selecting suitable images, from for example, the Landsat archive, panchromatic 15m images. The required input then is two multi-temporal images, and a stack of shapefiles for denoting glaciers and terrain assumed to be stable (i.e. no ocean, lakes, glaciers etc.). The pre-processing entails the importation, coarse scale matching procedures over and co-registration of the two images as well as generating binary masks. The main processing includes the multi-algorithm fine scale matching procedures over glaciers. The post-processing includes the preparation and adjustment of the displacements, outlier filtering and preparation of the quality indicator(s) mask, and summary statistics off/on glaciers. Output data include the raw level 1a displacements and level 1b (filtered displacements) multilayer data (in Geotif format) and header information.

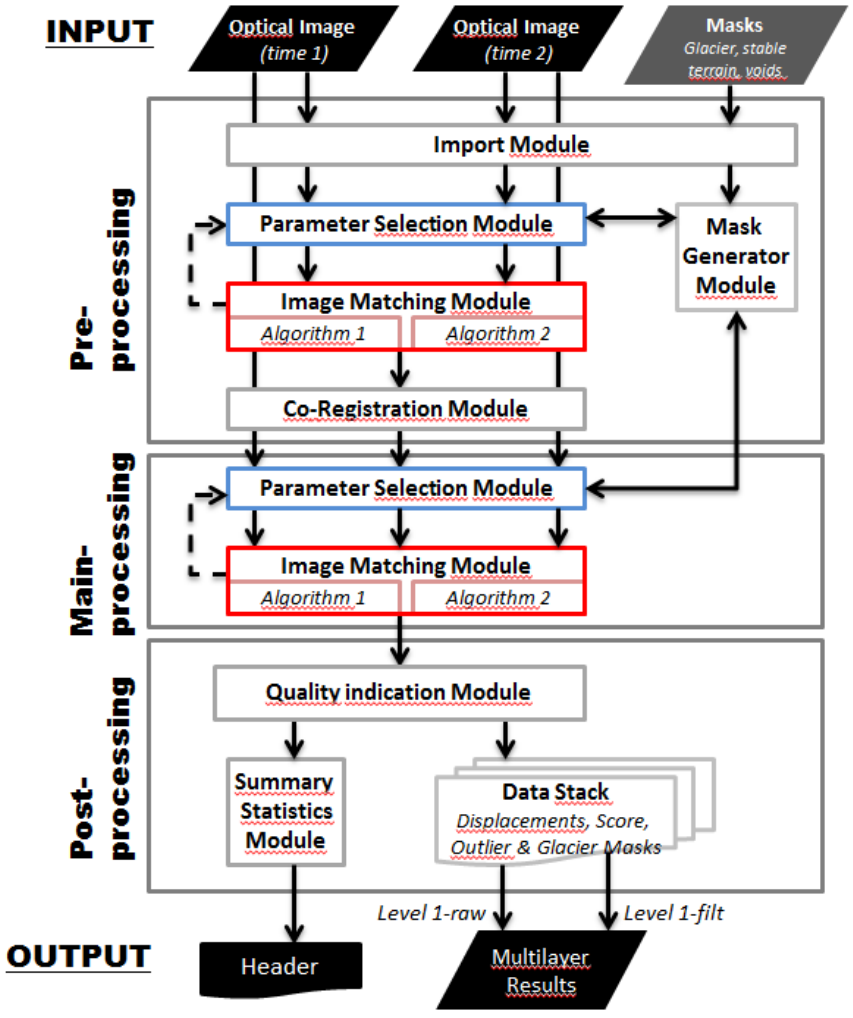

Fig. 6.1: Structure of the glacier velocity from optical data macro module.

6.2 Module description

6.2.1 Import module

This module determines from the user which files should be input into the processing chain including both image names and file locations and shapefile names and file locations. Shapefiles are those of the glacier areas determined from the Glaciers_cci area product (Section 3) or from the GLIMS database.

Input variables:

- As input by user through a GUI, or as text file. Import module is flexible. At a minimum the file locations and names of the images and shapefiles are required.

Output variables:

- *vec_x1, vec_ y1, mtx_im1*
- *vec_x2, vec_ y2, mtx_im2*
- *mtx_score1, mtx_void1, mtx_score2, mtx_void2*
- *const_time1, const_time2*
- *glacier_shape*

6.2.2 Parameter selection module

General description and definition of the objects to be computed

This module determines manually or (semi-)automatically the window sizes that the algorithm will use to match the multi-temporal images, and the matching grid. As an alternative to the matching grid, a list of locations to be matched can be imported (e.g. profiles). These parameters will be algorithm dependent, and multiple parameters may be required depending upon how many and which algorithms are implemented. The values in this module may be set standard for different input data (i.e. Landsat, ASTER, Sentinel) and regions. Some parameters such as *const_search_window_size* are undefined for Fourier methods. This module has two processing modes: a pre-processing and a main-processing mode. In the pre-processing mode, the window sizes and grids may be coarser than within the main-processing mode. In the main processing mode, the co-registration displacements are taken into consideration and passed through also to the image processing module.

Definition of variables

Input variables:

- *vec_x1,vec_ y1, mtx_im1*: image 1
- *vec_x2,vec_ y2, mtx_im2*: image 2

Output variables:

- *vec_x_grid, vec_y_grid* OR *vec_x_pts, vec_y_pts* : locations to be matched
- *const_template_widow_size*: template window size in image 1
- *const_search_window_size*: search window size in image 2

6.2.3 Mask generator module

General description

This generates binary mask grid files that are consistent with the pixel size and grid of the input DEM. Specifically, it determines whether a pixel of the image lies within input polygons. Two separate binary masks should equal 1 for pixels that are glacier and stable terrain respectively, to be used in the co-registration module, quality indicator module and the summary statistics module.

Definition and detailed description of the objects to be computed

The objects to compute in this module are two separate binary masks in SAR geometry for pixels that are glacier respectively stable terrain.

Definition of variables

Input variables:

- *Shapefiles (glaciers, clouds, ocean, lakes, voids, etc)*
- *vec_x1 (master), vec_y1 (master*

Output variables:

- *mask_glac: Binary mask equal to 1 for glaciers*
- *mask_terr: Binary mask equal to 1 for stable terrain*

Model equations and logical flow diagram

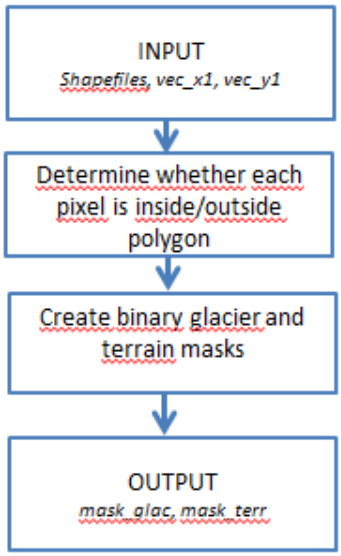

Fig. 6.2: Logical flow diagram of the mask generator module.

Pseudo Code

1. *Loop through each shapefile*
2. *Loop through each polygon object in shapefile.*
3. *Loop through each pixel coordinate in master image.*

4. *For each pixel of master image, determine whether pixel is inside or outside glacier polygon. Set mask_glac == 1 for inside, and 0 = outside to mask.For mask_terr, set 1 for outside glacier mask and 0 for inside the glacier mask.*

6.2.4 Image matching module

General description and definition of the objects to be computed

Use the parameters from the *parameter selection module* to image match the multi-temporal images to determine displacement vectors between the two images. This module has the flexibility to apply multiple algorithms. The output are displacements in the x and y directions between each of the input images at a resolution of the template window size. A score or correlation layer is also provided for each displacement.

Definition of variables

Input variables:

- *vec_x1,vec_ x2, mtx_im1*: image 1
- *vec_y1,vec_y2, mtx_im2*: image 2
- *vec_x_grid, vec_y_grid* OR *vec_x_pts, vec_y_pts*
- *const_template_widow_size*
- *const_search_window_size*
- *mask_terr, mask_glac*
- *[optional] const_xcoreg, const_ycoreg*

Output variables:

- *mtx_dx1 / mtx_dy1* : displacement in x/y-direction from slave image (image 2) to master image (image 1).
- *mtx_score1 :* A quality indicator depending upon the algorithm used. May be a correlation value, or a signal to noise ratio (SNR), for example.
- *mtx_dx*, mtx_dy*,mtx_score* :* where * represents a number counter for each algorithm implemented

Model equations and logical flow diagram

The two model equations are shown in the flow diagram.

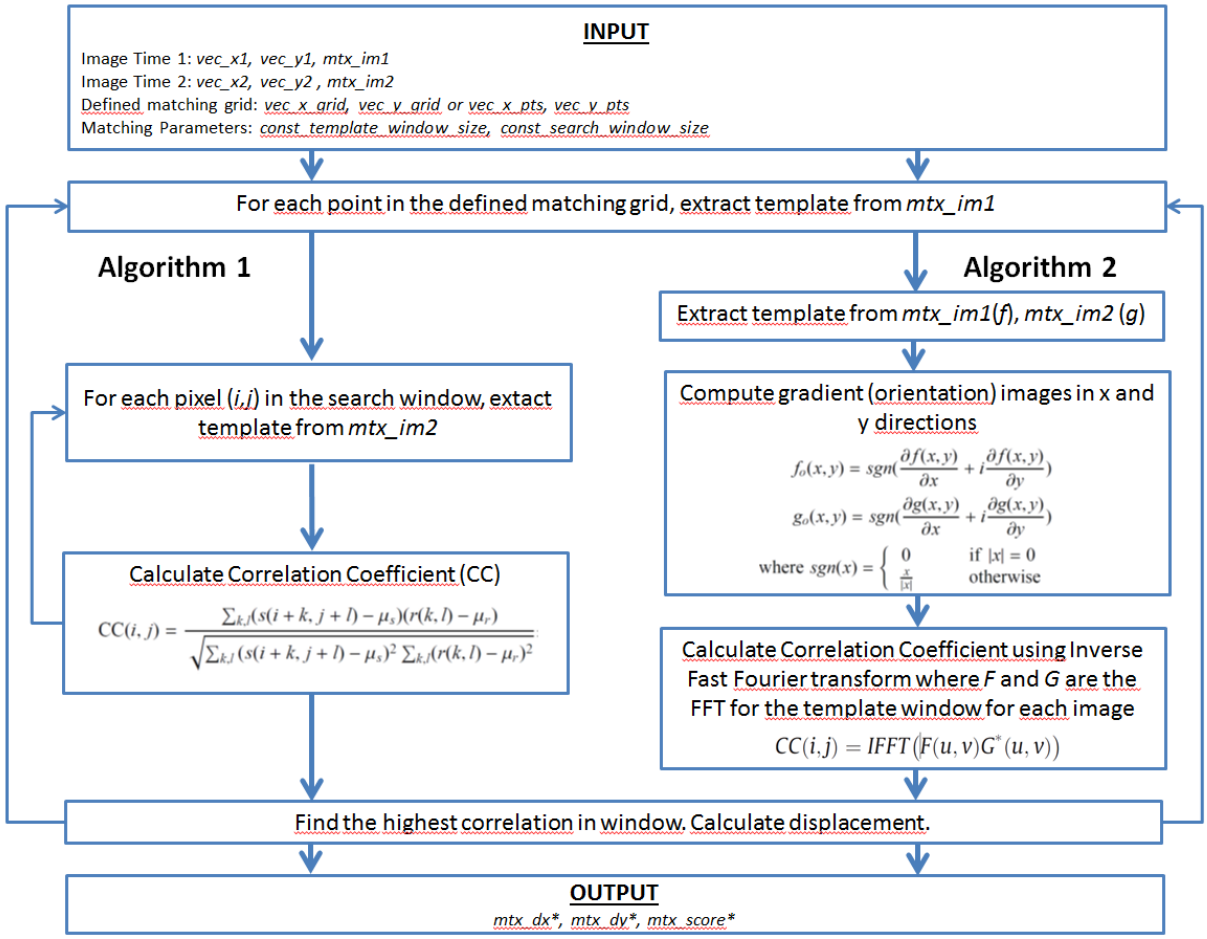

Fig. 6.3: Logical flow diagram of the image matching module.

Pseudo Code

1. *The image matching process code vary for different algorithms.*
2. *Loop through all input matching locations (vec_x_grid, vec_y_grid OR vec_x_pts, vec_y_pts)*
3. *For brute-force algorithm, within each iteration, template from image 1 is extracted, and run algorithmby looping over the search window size. Find highest score/correlation.*
4. *For Fourier methods, the search in image 2 is not required. The correlation coefficient is determined by multiplying the Fast Fourier Transform of the orientation images from both times and reverting using the Inverse Fast Fourier Transform. Find highest score/correlation.*
5. *If accessed in main-processing mode, then all displacements are corrected for the co-registration parameters.*

6.2.5 Co-registration module

General description and definition of the objects to be computed

This routine uses the displacements derived in the image matching module over stable terrain to determine the two translational co-registration parameters. The outputs from the module are two constants that will be used to adjust all displacements after the final image matching is performed.

Definition of variables

Input variables:

- *mtx_dx** , *mtx_dy**, *mtx_score**
- *mask_terr:* where * is a number reflecting the number of algorithms implemented

Output variables:

- *const_xcoreg*: mean x-direction displacement over stable terrain
- *const_ycoreg*: mean y-direction displacement over stable terrain

Model equations and logical flow diagram

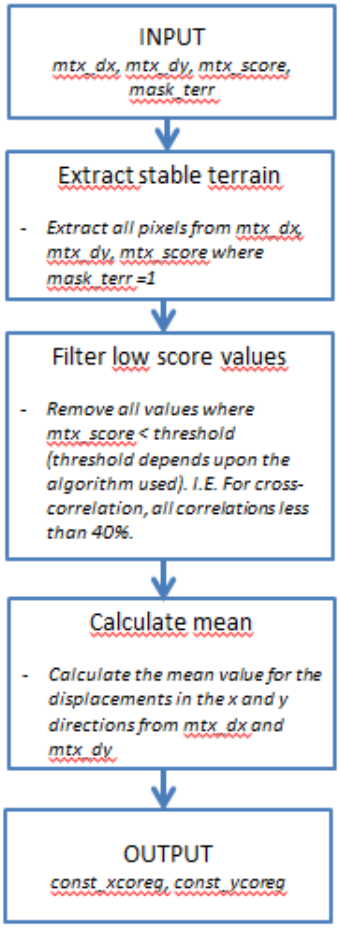

Fig. 6.4: Logical flow diagram of the co-registration module.

Pseudo Code

1. *Extract all displacements over stable terrain.*
2. *Filter outliers. If mtx_score < threshold, then remove. Threshold varies depending upon algorithm used. For cross correlation, a typical value will be 40%. If Signal-to-Noise ratio is being used, then the threshold may be data specific.*
3. *Calculate the mean x-direction and y-direction displacement.*

6.2.6 Quality indication module

General description and definition of the objects to be computed

This module implements a number of post processing routines to derive a quality indicator mask that is output with the displacements. This includes thresholding on the score matrix, results from a low-pass and other smart- filters, multi-algorithm comparisons.

Definition of variables

Input variables:

- *mtx_dx*, mtx_dy*,mtx_score*:* where * represents each algorithm implemented
- *mask_terr, mask_glac*

Output variables:

- *mtx_qual_ind1**: Correlation value from matching algorithm.
- *mtx_qual_ind2**: Signal to Noise Ratio (SNR)
- *mtx_dx*, mtx_dy*, mtx_score*:* where * represents each algorithm implemented
- *mask_terr, mask_glac*

Model equations and logical flow diagram

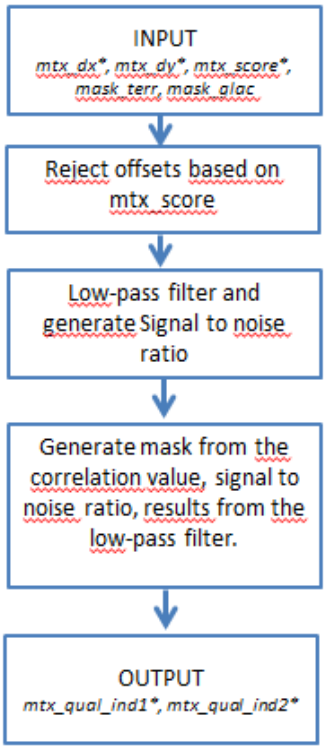

Fig. 6.5: Logical flow diagram of the quality indication module.

Pseudo Code

1. Read the displacement offset fields in x and y directions
2. Read the quality indicator field
3. Threshold on the score matrix (e.g. peak signal-to-noise ratio larger than 6.0)
4. Low-pass (or other) filter displacement field
5. Compute difference of filtered and original displacement field and determine outliers from which to add to the quality mask
6. Generate quality mask (mtx_qual_ind**) for displacement estimates

6.2.7 Data stack module

General description and definition of the objects to be computed
This forms the basis for the output. It merges the displacement matrices and the quality indicator matrices, indicator masks (glaciers and stable terrain), etc. into one coherent data stack (data cube, multi-layer grid etc.). Since all the different layers will be in a consistent grid, they can be merged into one file. If cases of multiple algorithms, then all results should be merged into one consistent grid, or multiple multilayer outputs will be created.

Definition of variables

Input variables:

- *mtx_qual_ind1*: Correlation value
- *mtx_qual_ind2*: Signal to Noise Ratio (SNR)
- *mtx_dx*, mtx_dy*,mtx_score*:* where * represents each algorithm implemented
  mask_terr, mask_glac

Output variables (geotiff):

- *Level 1-raw: Raw Displacements*
- *Level 1-filt: Filtered Displacements*

Model equations and logical flow diagram
All output grids are merged into one file for export.

7. Velocity from microwave sensors

7.1 General macro-module description

The macro-module for SAR images begins with an optional module for selecting suitable images, from for example ENVISAT ASAR, ALOS PALSAR or TerraSAR-X. The required input data are two multi-temporal images with associated headers, and optionally a stack of shapefiles for denoting glaciers and terrain assumed to be stable (i.e. no ocean, lakes, glaciers etc.) and a Digital Elevation Model (DEM). The pre-processing entails data import, coarse scale matching procedures, co-registration of the two images as well as generating the optional binary masks or the optional geocoding. The main processing includes the multi-algorithm fine scale matching procedures, if selected only over glaciers. The post-processing includes the preparation and adjustment of the displacements, outlier filtering and preparation of the quality indicator(s) mask, summary statistics off/on glaciers, and data export. Output data include multilayer data (in geotiff or NetCDF) and header information.

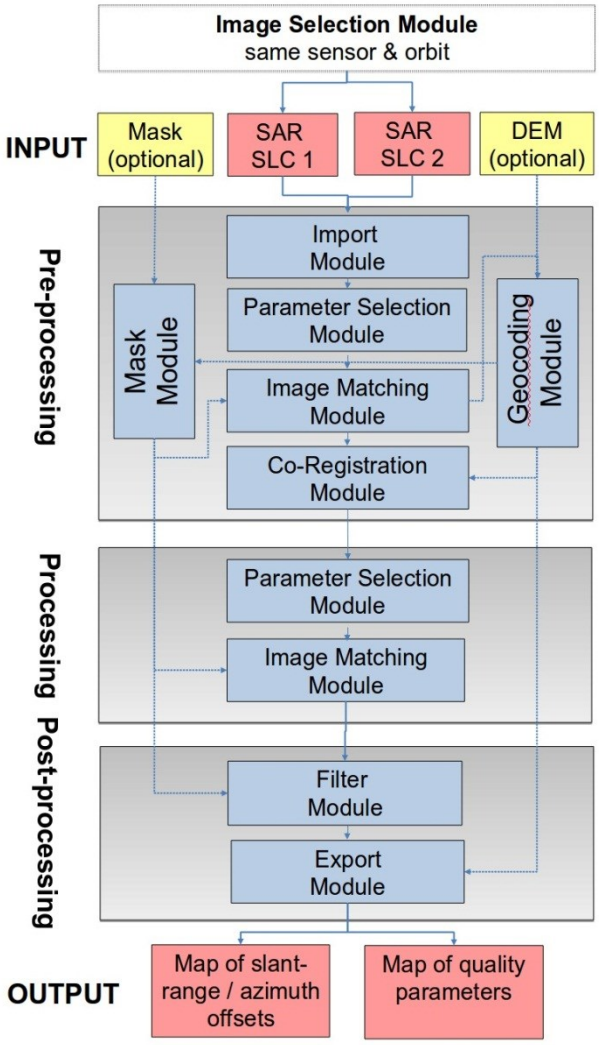

Fig. 7.1: Structure of the data macro module for glacier velocity from radar.

7.2 Module description

7.2.1 Parameter selection module

General description

This module determines manually or (semi-)automatically the window sizes that the algorithm will use to match the multi-temporal images and the matching grid. The values in this module may be set standard for different input data and regions. The module has two processing modes: a pre-processing and a main-processing mode. In the pre-processing mode, the window sizes and grids may be coarser than within the main-processing mode. In the main processing mode, the co-registration displacements are taken into consideration via a registration offset polynomial and passed through to the image processing module. A mask with glaciers, stable areas and other features (e.g. sea) can be included to focus the matching on stable areas during the pre-processing step (i.e. for the coarse image co-registration) and on glaciers during the main-processing step. In general, a regular sampling of matching positions is considered. If a mask is available, certain locations will be ignored.

Definition and detailed description of the objects to be computed

In this module the locations to be matched, the template window sizes in image 1 and 2, and the oversampling factor of the input images to be applied during matching.

Definition of variables

Input variables:

- *par_1, par_2:* acquisition parameters for image 1 and 2
- *poly_x, poly_y* : registration offset polynomial in x and ycoordinates
- *mask_glac_sar*: binary mask equal to 1 for glaciers in SAR geometry (optional)
- *mask_terr_sar*: binary mask equal to 1 for stable terrain in SAR geometry (optional)

Output variables: (all written to single ASCII file off_par)

- *vec_x_grid, vc_y_grid* : locations to be matched
- *const_template_widow_size*: template window size in image 1
- *const_search_window_size*: search window size in image 2
- *ovr_fact*: oversampling factor or input images to be applied during matching

Model equations and logical flow diagram

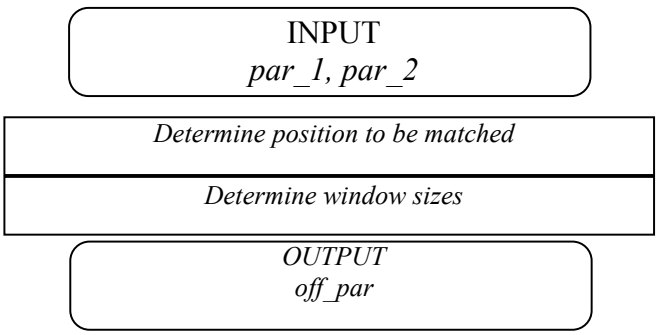

Fig. 7.2: Logical flow diagram of the parameter selection module.

Pseudo Code

1. read parameters of input images
2. determine starting range pixel, ending range pixel, step in range pixels, starting azimuth line, ending azimuth line, step in azimuth pixels in master image
3. optionally read registration offset polynomials in x and y coordinate
4. determine positions to be matched in slave image from positions in master image and offset polynomials
5. optionally read glacier mask or stable terrain mask transformed in SAR geometry
6. optionally mask positions to be matched only on stable terrain (coarse registration) or on glaciers (main processing)
7. determine template window sizes
8. determine oversampling factor
9. create parameter file including the output variables

7.2.2 Geocoding module

General description
Generate lookup tables for geocoding of SAR image to map geometry and for inverse geocoding of geographic information in map geometry into SAR geometry. This requires an image-matching module for refinement. Geocoding of SAR images is acknowledged as an automatic procedure that is already implemented in various software packages. Only a general logical flow diagram is thus reported here (no pseudo code).

Definition and detailed description of the objects to be computed
The objects to compute in this module are lookup tables for the transformation of images from the SAR to the map geometry and for the map to the SAR geometry.

Definition of variables

Input variables:
- *dem*: digital elevation model
- *par_1:* acquisition parameters for image 1
- *slc_1: 2*-D matrix of single-look complex values for image 1

Internal variables:
- *sar_sim: 2-D matrix of single-look intensity values for image 1 simulated from dem*

Output variables:
- *map_to_sar*: inverse geocoding lookup table
- *sar_to_map*: geocoding lookup table

Model equations and logical flow diagram

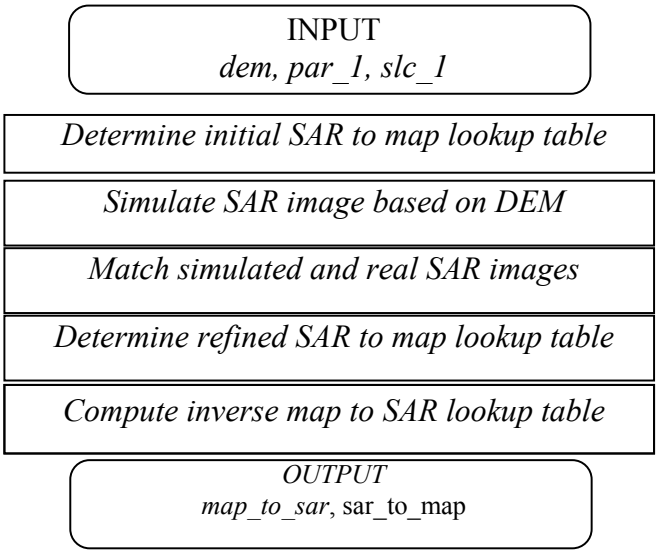

Fig. 7.3: Logical flow diagram of the geocoding module.

7.2.3 Mask generator module

see 6.2.3.

7.2.4 Image matching module

General description
Use the parameters from the *parameter selection module* to calculate an image of the displacement in m/day by locally matching the two SAR repeat pass images by applying incoherent cross correlation.

Definition and detailed description of the objects to be computed
The module calculates images of the local displacement in range and azimuth direction given in metre per day, and the corresponding crosscorrelation coefficient as a measure for the quality of the displacement.

Definition of variables

Input variables:

- *Slc-1, Slc-2*
- *Wrapping function poly-x and poly-y*
- *mtx_dx1, mtx_dy1*
- *Range and azimuth pixel size (in metre)*
- *Time interval between slc-1 and slc-2 acuisitions (in days)*

Internal variables:

- Ampl-1 amplitude image calculated from slc-1

- Ampl-2, amplitude image from slc-2 resampled using wrapping function poly-x and poly-y
- TW_ampl-1 – template window extracted from ampl-1 image, matrix of size [const_template_window_size, const_template_window_size]
- TW_ampl-2 – template window extracted from ampl-2 image, matrix of size [const_template_window_size, const_template_window_size]
- FFT_TW_ampl-1 Fast Fourier Transform of TW_ampl-1,matrix of size [2*const_template_window_size, 2*const_template_window_size]
- FFT_TW_ampl-2 Fast Fourier Transform of TW_ampl-2 [2*const_template_window_size, 2*const_template_window_size]
- R cross correlation matrix of size[2*const_template_window_size, 2*const_template_window_size]

Output variables:

- DISPL: image of displacement in range and azimuth, size [mtx_dx1, mty_dx1]
- RIMG: image of maximum crosscorrelation coefficiet for displacements, size [mtx_dx1, mty_dx1]
- mtx_score: quality indicator (e.g. correlation value or a signal to noise ratio (SNR)

Model equations and logical flow diagram

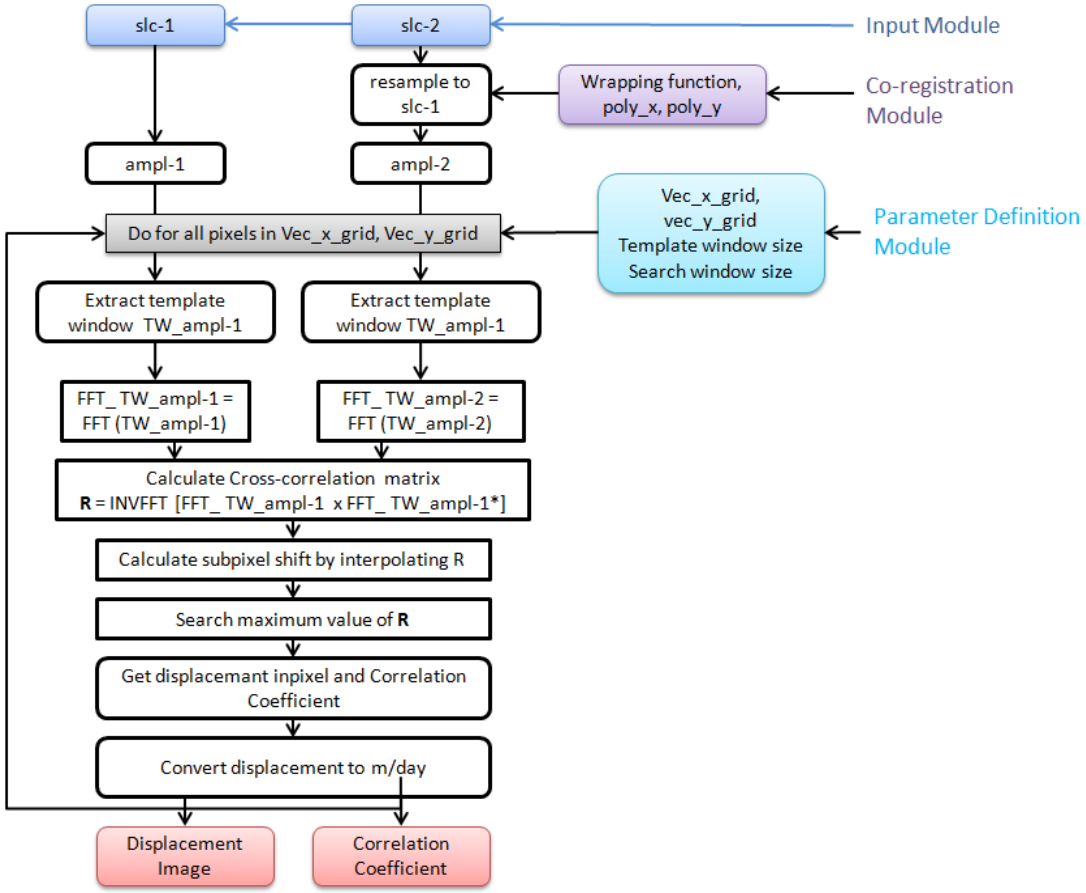

Fig. 7.4: Logical flow diagram of the image matching module

Pseudo Code

1. *Calculate amplitude image ampl-1 from slc-1*
2. *Resample slc-2 to slc-1 by applying wrapping function poly_x and poly_y and calculate amplitude image ampl-2 from slc-2*
3. *Calculate displacement at pixels given in fields vec_x_grid, vec_y_grid by applying the following steps:*
   Do for all pixels in ampl-1 image:
4. *Extract template window from ampl-1 centred around matching centre point (x,y), TW_ampl-1size*
5. *Extract template window from ampl-2 around centre point (x,y); TW_ampl-2*
6. *Make template window of TW_{ampl-1} zero-mean: Calculate mean value over template window of ampl-1 and subtract mean from pixel values of data*
7. *Make template window of TW_{ampl-2} zero-mean Calculate mean value over template window of ampl-2 and subtract mean from pixel values of data*
8. *Apply FFT to template TW_ampl-1: $FFT\_TW_{ampl-1} = FFT(TW\_ampl-1)$*
9. *Apply FFT to TW_ampl-2: $FFT\_TW_{ampl-2} = FFT(TW\_ampl-2)$*
10. *Calculate Cross-correlation matrix by*
    ***R** = IFFT[FFT_ TW_ampl-1 x FFT_ TW_ampl-2*]*
    *where * denote the complex conjugate, IFFT () is the Inverse Fast Fourier Transform.*
    ***R** is the cross-correlation matrix*
11. *Search for the maximum correlation coefficient in matrix **R***
12. *Calculate sub-pixel shift by interpolation of R around the maximum value of R, using 2-dimensional sinc-interpolation*
13. *Extract shift in pixel from location of maximum correlation coefficient in matrix R*
14. *Convert pixel shift to m/day by applying the range and azimuth pixel size and dividing by the time interval of the acquisition of slc-1 and slc-2*

7.2.5 Co-registration module

General description

This routine will use the displacements mtx_dx1 and mtx_dy1 derived in the image matching module, optionally masked over stable terrain only, to determine the two translational co-registration parameters.

Definition and detailed description of the objects to be computed

The objective is to compute are the registration offset polynomials in x and y coordinates. The 4 order polynomials used to model the range and azimuth offsets are of the type:
range_offset = A0 + A1*range + A2*azimuth + A3*range*azimuth
azimuth_offset = B0 + B1*range + B2*azimuth + B3*range*azimuth

Definition of variables

Input variables:

- *mtx_dx1 / mtx_dy1* : displacement in x/y-direction from master image (image 1) to slave image (image 2).
- *mtx_score: quality indicator (e.g. correlation value or a signal to noise ratio (SNR))*

Output variables:

- *poly_x*: registration offset polynomial in x coordinate (see definition before);
- *poly_y*: registration offset polynomial in y coordinate (see definition before).

Model equations and logical flow diagram

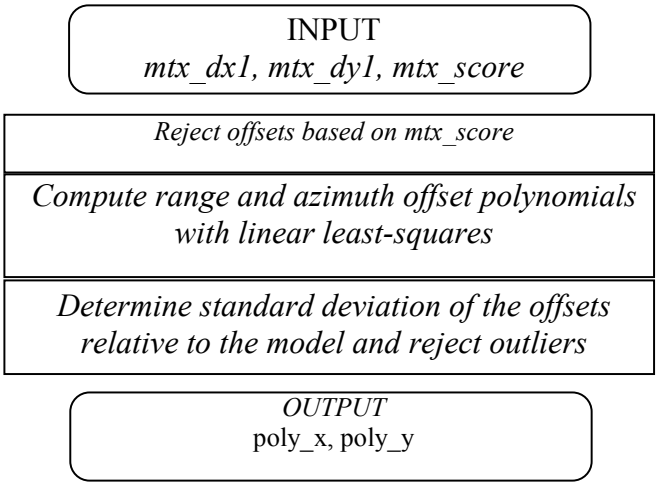

Fig. 7.5: Logic flow diagram of the co-registration module

Pseudo Code

1. *Read the displacement offset fields in x and y directions*
2. *Read the quality indicator field*
3. *Reject offset estimates with a quality indicator below an indicated threshold*
4. *Use linear least-squares to compute range and azimuth registration offset polynomials from offsets estimated by image matching module in the form of range_offset = A0 + A1*range + A2*azimuth + A3*range*azimuth azimuth_offset = B0 + B1*range + B2*azimuth + B3*range*azimuth*
5. *Compute the deviation of the offset measurement away from the fit*
6. *Reject values above a certain thresholding*
7. *Iteratively repeat steps 4-6 with culled data until there is no further improvement in the standard deviation of the offsets relative to the model*
8. *Export registration offset polynomials*

7.2.6 Filter module

General description

This module implements a number of post processing routines to derive a quality indicator mask that is output with the displacements. This includes thresholding on the score matrix, results from a low-pass and other smart-filters, multi-algorithm comparisons.

Definition and detailed description of the objects to be computed

The object to compute in this module is a binary mask in SAR geometry equal to 1 for regions to consider displacement estimates and 0 for regions to discard displacement estimates.

Definition of variables

Input variables:

- *mtx_dx1* : displacement in x-direction from master image (image 1) to slave image (image 2)
- *mtx_dy1:* displacement in y-direction from master image (image 1) to slave image (image 2)
- *mtx_score:* quality indicator (e.g. correlation value or a signal to noise ratio (SNR))
- *mask_glac_sar*: binary mask equal to 1 for glaciers in SAR geometry
- *mask_terr_sar*: binary mask equal to 1 for stable terrain in SAR geometry

Output variables:

- *mask_filt_sar:* binary mask in SAR geometry equal to 1 for regions to consider displacement estimates and 0 for regions to discard displacement estimates

Model equations and logical flow diagram

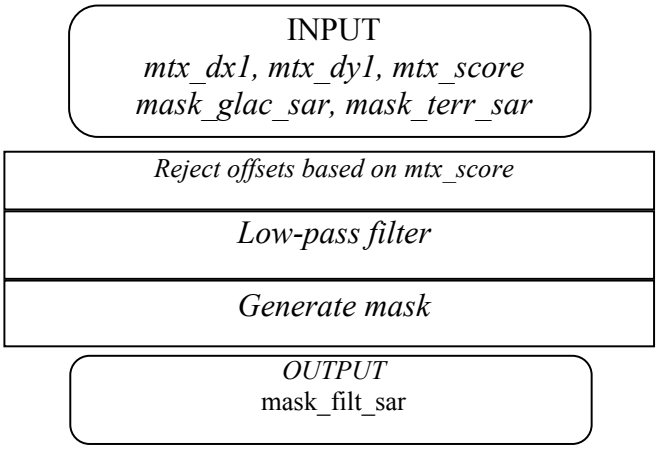

Fig. 7.6: Logical flow diagram of the filter module

Pseudo Code

1. *Read the displacement offset fields in x and y directions*
2. *Read the quality indicator field*
3. *Threshold on the score matrix (e.g. peak signal-to-noise ratio larger than 6.0)*
4. *Low-pass (or other) filter displacement field*
5. *Compute difference of filtered and original displacement field and remove outliers*
6. *Read binary masks of glaciers and terrain in SAR geometry*
7. *Generate mask to consider or discard displacement estimates outside glaciers*
8. *Combine rejection masks based on quality indicator, filter and glaciers into one offset mask in SAR geometry for export*

8. References

Abramowitz and Stegun (1972): Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables, New York: Dover Publications.

Bolch, T., B. Menounos and R.D. Wheate (2010): Landsat-based inventory of glaciers in western Canada, 1985 - 2005. Remote Sensing of Environment, 114(1), 127-137.

Draper, N. R. and Smith, H. (1981). Applied Regression Analysis, John Wiley & Sons, Inc.

Glaciers_cci (2012a): Algorithm Theoretical Basis Document version 1 (ATBDv1): Prepared by the Glaciers_cci consortium, 95 pp.

Glaciers_cci (2012b): Product Validation and Algorithm Selection Report (PVASR): Prepared by the Glaciers_cci consortium, 113 pp.

Glaciers_cci (2013): Input Output Data Definition (IODD): Prepared by the Glaciers_cci consortium, 25 pp.

Paul, F. (2007): The New Swiss Glacier Inventory 2000 - Application of Remote Sensing and GIS. Schriftenreihe Physische Geographie, Universität Zürich, 52, 210 pp.

Paul, F. and Kääb, A. (2005): Perspectives on the production of a glacier inventory from multispectral satellite data in the Canadian Arctic: Cumberland Peninsula, Baffin Island. Annals of Glaciology, 42, 59-66.

Rinne, E.J., Shepherd A., Palmer S., van den Broeke M.R., Muir A., Ettema J. & Wingham D. (2011): On the recent elevation changes at the Flade Isblink Ice Cap, northern Greenland. Journal of Geophysical Research, 116, F03024, doi:10.1029/2011JF001972.

Sandwell, D. T. (1987): Biharmonic spline interpolation of GEOS-3 and SEASAT altimetry data. Geophysical Research Letters, 14 (2), 139-142.

Sheppard, W. F. (1903): New Tables of the Probability Integral. Biometrika, 2, 174-190 (available at http://biomet.oxfordjournals.org/content/2/2/174.full.pdf).

Sheppard, W. F.(1939): The Probability Integral. Cambridge, University Press.

Abbreviations

ALOS	Advanced Land Observing Satellite
ASAR	Advanced Synthetic Aperture Radar
ASTER	Advanced Spaceborne Thermal Emission and Reflection radiometer
CC	Correlation Coefficient
CCI	Climate Change Initiative
CDED	Canadian Digital Elevation Dataset
CGIAR	Consultative Group on International Agricultural Research
CPU	Central Processing Unit
DARD	Data Access Requirements Document
DEM	Digital Elevation Model
DGPS	Differential GPS
dDEM	differential DEM
DLR	German Aerospace Centre
DP-RT-RepAltDEM	sloPe Repeat Track Repeat Altimetry
DS-RT-RepAltDEM	Subtracting Repeat Track Repeat Altimetry
DTED	Digital Terrain Elevation Data
ELA	Equilibrium Line Altitude
ERS	European Remote Sensing Satellite
ESA	European Space Agency
ETM+	Enhanced Thematic Mapper plus
FCDR	Fundamental Climate Data Record
FoG	Fluctuation of Glaciers
GCOS	Global Climate Observing System
GDB	GLIMS Data Base
GDEM	Global DEM
GIC	Glaciers and Icecaps
GIS	Geographic Information System
GLAS	Geoscience Laser Altimeter System
GLIMS	Global Land Ice Measurements from Space
GLS	Global Land Survey
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
GV	Glacier Velocities Product
ICESat	Ice, Cloud, and Elevation Satellite
ID	IDentification number
IGOS	Integrated Global Observing Strategy
InSAR	Interferometric SAR

L1T	Level 1 T (terrain corrected)
LDCM	Landsat Data Continuity Mission
LIDAR	LIght Detection And Ranging
LOS	Line of Sight
MAI	Multiple Aperature InSAR
NASA	National Aeronautic and Space Administration
NDSI	Normalized Difference Snow Index
PALSAR	Phased Array type L-band SAR
PSD	Product Specifications Document
PVP	Product Validation Plan
RA XO	Repeat Altimetry Cross-Over
RMSE	Root Mean Square Error
RP-RT-RepAlt	Rectangular Plane Repeat Track Repeat Altimetry
RR	Round Robin
SAR	Synthetic Aperture Radar
SD	Standard Deviation
SEC	Surface Elevation Change
SLC	Scan Line Corrector
SNR	Signal to Noise Ratio
SoW	Statement of Work
SPOT	System Pour l'Observation de la Terre
SRTM	Shuttle Radar Topography Mission
SSC	Slant range Single look Complex
SWIR	Short Wave InfraRed
TM	Thematic Mapper
TP-RT-RepAlt	Triangular Plane Repeat Track Repeat Altimetry
URD	User Requirements Document
USGS	United States Geological Survey
UTM	Universal Transverse Mercator
WGI	World Glacier Inventory
WGMS	World Glacier Monitoring Service

Add 7.2.7 Export Module at the very end.

General description
This module exports the final products in map geometry. The sar_map_geometry convertor from the geocoding module is applied to the outputs of the Image matching and filter modules.

Definition and detailed description of the objects to be computed
The object to export displacement maps in x- an y-direction from master image to slave image and the quality indicator map in map geometry.

Definition of variables
Input variables:
• mtx_dx1 : displacement in x-direction from master image (image 1) to slave image (image 2) in SAR geometry
• mtx_dy1: displacement in y-direction from master image (image 1) to slave image (image 2) in SAR geometry
• mtx_score: quality indicator (e.g. correlation value or a signal to noise ratio (SNR)) in SAR geometry
• mask_filt_sar: binary mask in SAR geometry equal to 1 for regions to consider displacement estimates and 0 for regions to discard displacement estimates

Output variables:
• mtx_dx1_map : displacement in x-direction from master image (image 1) to slave image (image 2) in map geometry
• mtx_dy1_map: displacement in y-direction from master image (image 1) to slave image (image 2) in map geometry
• mtx_score_map: quality indicator (e.g. correlation value or a signal to noise ratio (SNR)) in map geometry

Model equations and logical flow diagram
INPUT
mtx_dx1, mtx_dy1, mtx_score, mask_filt_sar
Mask
Geocoding
OUTPUT
mtx_dx1_map, mtx_dy1_map, mtx_score_map
Fig. 7.8: Logic flow diagram of the export module

Pseudo Code
The pseudo code of the module is here reported.
1. Read the displacement offset fields in x and y directions
2. Read the quality indicator field
3. Apply mask to image of displacement in range and azimuth and to quality indicator map
4. Geocode images of displacement in range and azimuth and of quality indicator to map geometry

5. Export final images

Import and Image Selection Module

General description
The module for selecting SAR images is optional, as manual selection of suitable image pairs should be also performed. The idea behind having such a module is to automatically select images to be matched right after their acquisition. Indeed, image matching based on SAR images has a high potential to be routinely implemented in a near-real time processing strategy, because it is highly automatic. This module should identify SAR images acquired along the same image geometry (i.e. same mode, incidence angle and polarization).

Definition and detailed description of the objects to be computed
There are no objects to be computed in this module, the module is designed to support automatic selection of suitable SAR image pairs in case of an operational mission.

dem differencing:

4.2.1 Import module

This module determines from the user which files should be input into the processing chain including both DEM names and locations and shapefile names and locations. Shapefiles at a minimum need to be those of the glacier areas as determined from the Glacier Area system (Section 3) or from the GLIMS database. Some DEMs also contain a score (correlation) mask that reveals good and bad values as well as a void mask (i.e. SRTM). Since these are not available for all products, they are just optional inputs.

Definition of variables

Input variables:

- As input by user through a GUI, or as text file (the import module is flexible).

Output variables:

- *vec_x1, vec_ y1, mtx_dem1*
- *vec_x2, vec_ y2, mtx_dem2*
- *mtx_score1, mtx_void1, mtx_score2, mtx_void2 (if available)*
- *const_time1, const_time2*
- *glacier_shape*